

A METHODOLOGICAL FRAMEWORK FOR DECISION-THEORETIC
ADAPTATION OF SOFTWARE INTERACTION AND ASSISTANCE

by

Bowen Hui

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Copyright © 2011 by Bowen Hui

Abstract

A Methodological Framework for Decision-Theoretic Adaptation of Software Interaction
and Assistance

Bowen Hui

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2011

In order to facilitate software interaction and increase user satisfaction, various research efforts have tackled the problem of software customization by modeling the user's goals, skills, and preferences. In this thesis, we focus on run-time solutions for adapting various interface and interaction aspects of software. From an intelligent agent's perspective, the system views this customization problem as a decision-theoretic planning problem under uncertainty about the user. We propose a methodological framework for developing intelligent software interaction and assistance. This framework has been instantiated in various case studies which are reviewed in the thesis. Through efforts of data collection experiments to learn model parameters, simulation experiments to assess system feasibility and adaptivity, and usability testing to assess user receptiveness, our case studies show that our approach can effectively carry out customizations according to different user preferences and adapt to changing preferences over time.

Dedication



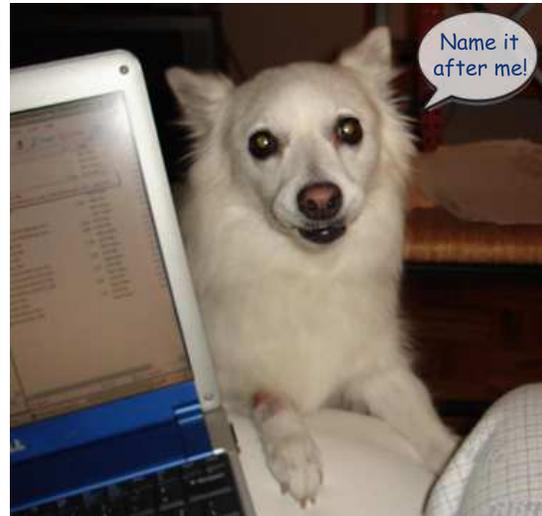
(a) Minnie



(b) Johnny



(c) Casper



(d) Daisy

Figure 1: How I made it all these years.

Acknowledgements

First and foremost, I would like to thank my advisor, Craig Boutilier, for all his patience and continual support in my work. After working together for so many years, it still amazes me that the hardest questions always come from him. I am very lucky to have had the opportunity to work with him. I would also like to thank my committee members, Ravin Balakrishnan, Rich Zemel, and John Mylopoulos, and my external examiner, Anthony Jameson, for all their questions, direction, and support. It's been very rewarding to work on a topic that is at the intersection of their interests.

The collaboration across different labs and projects has been the most enjoyable part of my degree: Think-and-Link project with John Mylopoulos and the University of Oregon, computer literacy project with Eileen Wood at Wilfrid Laurier University, stimulating conversations in Ravin's lab, intelligent psychiatric software with Gitte Lindgaard at Carleton University and Bill Winogron at S4Potential, and collaborations with Pourang Irani at the University of Manitoba.

Personally, I am in debt to my DGP friends, volleyball friends, and the Toronto-Greek friends (in no particular order): Mike McGuffin, Gonzalo Ramos, Winnie Tsang, Mike Wu, Abhishek Ranjan, Brad Reid, Tovi Grossman, Mike Neff, Joe Laszlo, Anastasia Bezerianos, Fanis Tsandilas, George Chalkiadakis, Meng Voong, Fanny Iq, Kelly Poupart, Haipo Yang, Julie Chun, Chris Quach, Miki Cheung, Sue Cheung, Yun Chan, Dave Yen, Imelda Nuwisah.

My main source of support over the last few years comes from Minnie, Johnny, Daisy, Casper, and Brad. Without your patience and smiles, I would have never pulled through.

Contents

1	Introduction	1
1.1	Research Objectives	4
1.2	List of Contributions	7
1.3	Outline of Thesis	9
2	Background	11
2.1	Bayesian Networks and Dynamic Bayesian Networks	11
2.2	Markov Decision Processes	16
2.3	Influence Diagrams and Partially Observable Markov Decision Processes	20
2.4	Applications for Intelligent Software Adaptation	24
2.5	Challenges	34
2.5.1	Designing the Utility Function	35
2.5.2	Subjective Utilities: Modeling and Elicitation	36
2.5.3	Partial Observability Coupled with Long Term Policies	37
2.5.4	Continual Learning of Model Parameters	37
3	A DT Framework for Intelligent Customization	39
3.1	Developing the Decision-Theoretic Model	40
3.1.1	Interaction Scenario	41
3.1.2	POMDP-DAISI	44
3.1.3	Cross-Episodic Interaction	48

3.1.4	Solution Strategy	49
3.2	The DAISI Framework	50
3.3	Relevance to Research Objectives	53
4	Modeling Individual Differences	56
4.1	Do People Have Different Preferences?	56
4.2	Relevant User Characteristics	59
4.3	Characteristics that Influence Savings	63
4.3.1	A Bayesian User Model	64
4.3.2	Test Application and Context of Use	70
4.3.3	Simulation Results	75
4.3.4	Learning Model Parameters	81
4.3.5	Usability Experiments	87
4.4	Characteristics that Influence Disruption	90
4.4.1	Properties of Mental Models	91
4.4.2	A Mental Model of Function Location	93
4.4.3	Decision-Theoretic Action Selection	99
4.4.4	Simulation Experiments	102
4.4.5	Learning Mental Model Dynamics	107
4.4.6	Learning the Cost of Disruption	111
4.4.7	Usability Experiment	115
4.5	Summary	119
5	Personalized Goal Recognition	121
5.1	Classes of Goals	125
5.1.1	Highlighting	126
5.1.2	Simple Mathematical References	129
5.1.3	Edit All Titles	132

5.1.4	Initial Indentation	133
5.1.5	Re-Alignment	134
5.1.6	Multiple Node-Arrow Connection	136
5.1.7	Typing Words	137
5.2	Finite State Automata	138
5.2.1	Model Set-Up	138
5.2.2	Event Vocabulary	142
5.2.3	Highlighting Templates and Machines	148
5.3	Inferring the User's Current Goal	163
5.3.1	The Inference Task	163
5.3.2	Empirically Updating the Observation Model	165
5.3.3	Empirically Updating the Episode Prior	165
5.4	Simulation Experiments	166
5.4.1	System Set-Up	167
5.4.2	Simulated User for Executing Highlighting Goals	173
5.4.3	Comparison System Policies	174
5.4.4	Evaluation Metrics	175
5.4.5	Results	177
5.5	Usability Experiments	190
5.5.1	Experiment Set-Up	190
5.5.2	Results	194
5.6	Summary	199
6	Utility of Intelligent Assistance	202
6.1	Identifying Interaction Factors	205
6.2	Utility Structure	209
6.3	Adopting Models for Processing, Selection, and Savings	213
6.3.1	The Cost of Information Processing	213

6.3.2	The Cost of Target Selection	215
6.3.3	The Benefits of Savings	218
6.4	The Cost of Disruption	220
6.5	The Cost of Visual Occlusion	221
6.5.1	Experiments for Learning Occlusion	222
6.6	The Cost of Interface Bloat	226
6.6.1	Experiment for Learning Bloat	227
6.6.2	Simulation Experiment	230
6.7	Eliciting the Subjective Value of Help	239
6.7.1	Highlighting Task in the Customization Domain	240
6.7.2	The Value and Costs of Suggestions	242
6.7.3	Brief Review of Preference Elicitation	243
6.7.4	Experiential Elicitation for Interface Customization	246
6.7.5	Conceptual versus Experiential Elicitation	247
6.7.6	Ways to Improve the Experiential Elicitation Procedure	252
6.8	Summary	254
7	Conclusions and Future Work	256
7.1	Developing the User Model	257
7.2	Policy Design	259
7.3	Evaluation Approaches	262
7.4	Future Work	263
	Bibliography	268
	A Materials	286
A.1	Chapter 4 Questionnaire for User Types	286

B	Parameter Specification	288
B.1	Chapter 4 Case Study One Parameters	288
B.2	Chapter 5	300
B.3	Chapter 6 Case Study Two	307
C	Full Results	309
C.1	Chapter 4 Case Study One Results	309
C.2	Chapter 4 Case Study One Last M% Results	309

Chapter 1

Introduction

Software development has historically adopted a “one-size-fits-all” model in which applications are designed with a single target user group in mind, rather than tailoring the application functionality or interface to the needs of specific users. The ability to customize software has become increasingly important as users are faced with larger, more complex applications. For a variety of reasons, software must be tailored to specific individuals and circumstances [HLM03]. For example, different users may require different functionality from multi-purpose software [BCM04], prefer different modes of interaction, or use software on a variety of hardware devices [GW04]. Because of this complexity, online and automated help systems are becoming increasingly prevalent in helping users identify and master different software functions [HBH⁺98]. Such systems should ideally adapt the help they provide and the decision to interrupt [HA03] to account for specific user preferences.

As such, researchers have recognized that different people have varying levels of expertise, different preferences, needs and goals in using software, thus, requiring different kinds of products and services to accommodate various circumstances. In this context, we adopt a broad perspective of *software customization* that adapts the availability of an application’s functionality, its navigation structure, the presentation layout of infor-

mation, or the interactivity level with users. For example, automatic capitalization of sentence initial characters may be made available or not, certain menu items may be shown, hidden, or placed in different parts of the interface, pop-up boxes may vary in transparency levels, and text prediction may be tuned to only make the predictions when the system's "quality" estimate is high. The main areas of research in Computer Science that tackles the problem of software customization are Requirements Engineering (RE), Human-Computer Interaction (HCI), and Artificial Intelligence (AI). For example, RE researchers are interested in defining the software functionality as the design space for a specific type of application and then partitioning that space into different product families [KKLL99] or traversing that space to identify a best match based on a pre-specified individual user profile [HLM03]. These approaches are customizations that occur at *design time*. Once it has been defined and deployed, the software product does not change until another design iteration occurs. Thus, as the user's goals, preferences, and skills change over time, the software cannot keep up with these changes.

An approach commonly adopted by HCI researchers who are interested in this problem is to create *adaptable* software. Adaptable software refers to any application that provides users with an interface that allows users to specify the customization needed for the application (e.g., [MBB02]). Since users are able to manually customize the software even after the product has been designed and deployed, this approach provides a *run-time* solution to the software customization problem. The main advantage of adaptable software is that it leaves the user in complete control. At the same time, users who want any kind of customization done will need to be aware of their own preferences (so they know *what* to customize in the software) and to be familiar with the adaptable interface (so they know *how* to customize the software).

Since users are not necessarily aware of their own preferences in using the software and do not necessarily want to expend additional effort in using software, researchers in AI have proposed to automate the run-time software customization process using

adaptive systems. This perspective views the software as an *intelligent system*, whereby the software has an embedded autonomous agent with input, processing, and output abilities. For example, an adaptive system may monitor the history of user actions and infer certain patterns of behaviour or information about the user’s goals. This information can then be used to automatically change the interface (e.g., hide certain menu items) or create alternative uses of the application for the user (e.g., pop up a suggestion box such as “Did you know...?”). In addition to customizing software at the functionality level, adaptive systems have the added ability to tailor these functionalities more specifically to the user based on the observed interaction history. This problem is known as *software personalization*, which focuses on tailoring the data or the content of specific functionality in an application. For example, menu items may include user-specific macros, and text predictions may be tailored to model words and phrases that the user has written in the past. Overall, the main benefits of adaptive systems is that users no longer have to be aware of their own software preferences, users are not required to know which parts of the application are customizable, and the system may reveal information about the application that was not known to the user. However, with the uncertainty that arises in inferring user goals and user characteristics in specific domains and contexts, there is a risk of making a “wrong” customization that leads to the user losing trust in the system and rejecting the software altogether.

To reconcile these differences, recent efforts have focused on using *mixed initiative* approaches, where an adaptive system has the ability to initiate changes to the application (based on its beliefs about the user) as well as respond to user-initiated events (e.g., [Hor99b, Hor99a]). We do not pursue this debate between adaptable versus adaptive systems here. Rather, we adopt the mixed initiative approach by focusing on adaptive techniques that allow for adaptable user control. We refer to this problem as *intelligent software adaptation*. A major difficulty facing developers of such systems is the uncertainty associated with assessing the needs of a specific user. While hard-coded rules offer

some benefits, it is becoming apparent that probabilistic assessment of a user's needs based on observed behaviour offers considerable advantages [HBH⁺98, AZN99, GP00]. Following this direction, we approach the problem of intelligent software adaptation from a *decision-theoretic user modeling* perspective so that the agent learns a model of the user and uses that information to decide its course of actions during its interaction with the user. In this thesis, we focus on intelligently adapting software interaction and assistance, such as the location of functions (which in turn impacts ease of interaction), presentation attributes of interface widgets, the interactivity level when asking questions or making suggestions to users. We view this work as a way to help users interact with software more efficiently and seamlessly.

1.1 Research Objectives

Researchers have reported that a lack of established design guidelines exist for developing intelligent software adaptation systems [HGB07]. As such, the main contribution of this work is a methodological framework for developing intelligent software adaptation systems. We will identify the main components involved and show how the framework is used to build intelligent software adaptation systems through several case studies. The focus of our framework is the development of a decision-theoretic user model and the role it plays in balancing the tradeoffs among various interface and interaction criteria that are used in the system's intelligent decision making process.

Generally speaking, the objectives of intelligent software adaptation are to design software tailored to individuals so to minimize user effort and/or to maximize the ease of interaction during system-user interaction. In desktop applications, many kinds of system actions can be implemented to (potentially) achieve these objectives. Examples of automated system actions include the following: doing mundane work on the user's behalf (e.g., auto-completion), changing widget locations for more convenient access (e.g.,

moving or hiding them from the immediate interface), creating one-click shortcuts for common multi-step tasks (e.g., suggesting macros), changing the delivery of widgets (e.g., via animation), changing the presentation of widgets (e.g., level of transparency), sending reminders (e.g., hints or explanations in a text balloon), making suggestions (e.g., icons in a toolbar), asking questions explicitly (e.g., via a dialog box), etc. However, different actions have different consequences: an adaptive interface that hides unused menu items may be preferable for one user because it saves him from scanning unnecessary functions, but the same behaviour may be detrimental to other users who prefer to see all available functions (e.g., have high tolerance to bloat). With many types of users and situations involved, intelligent systems must be able to model a variety of circumstances and consequences so that appropriate tradeoffs can be made. Furthermore, system actions have effects beyond their immediate consequences. For example, a user who likes unused menu items hidden may find it annoying when he needs to use one of those functions in the future (i.e., cost of re-discovery). Our underlying assumption is that different people perceive these consequences differently; thus, the impact of the consequences varies across users. If this assumption is true, then intelligent systems need to be designed to learn individual differences that explain such differential impact and tailor the system's behaviour accordingly.

The general objectives of this work are to test this assumption in an interface customization setting, to develop ways to learn how users perceive automated assistance differently, and to leverage this knowledge in designing intelligent software adaptation systems that make appropriate tradeoffs when deciding the best course of action for a specific user. Specifically, we are interested in the following questions:

1. Modeling individual differences:

- (a) When presented with automated help, do (some) people use it? Under what conditions will people make use of automated help? E.g., only when the help is

a perfect match to what they want to do, or will a “partial” match be helpful? Will people use automated help in simple tasks? Will people use automated help that is not always available or not always presented in the same way, i.e., in adaptive systems?

- (b) All else being equal, what makes some people want to use adaptive help and others not? Can we propose a set of *user characteristics* to explain these kinds of individual differences? Can these characteristics vary within the same person over time? As a starting point, what user characteristics do the HCI and user modeling literature adopt for capturing individual users? How do these characteristics relate to the concepts of “user profiles”, “user groups”, and “user types” from the literature?
- (c) What kinds of behaviours are indicators for these characteristics? Can these behaviours be defined using event sequences acquired from standard input devices (i.e., without relying on eye tracking, gesture/image recognition, etc.)? How feasible is it to represent the relationship between these behaviours and characteristics using a stochastic model? Is it feasible to learn the parameters of such a stochastic model empirically from a sample participant pool?
- (d) Using the stochastic user model, can the characteristics be estimated online efficiently, so that users of a system employing the model would not notice a lag during task completion?

2. Modeling system actions and consequences in a decision-theoretic framework:

- (a) Since some people make use of help that is only a “partial” match to their intended goal, the system needs to quantify the *quality* of its automated help in order to know what “partial” means. Specifically, how should automated suggestions be quantified with respect to the user’s current goal? Also, how is this quality measure defined to account for cases where the system suggests

multiple actions of varying quality simultaneously (e.g., a toolbar with several icons)?

- (b) The HCI and user modeling literature has documented many interaction factors that make adaptive systems more or less usable for users. As well, the HCI literature also has proposed some design criteria to make software more usable in general. Among these interaction factors, which ones are relevant to the development of an intelligent software adaptation system, given the kinds of system help actions we are interested in?
- (c) Can predictive models of these interaction factors be formalized and constructed? What are the parameters of these models? Is it feasible to gather empirical data to learn these models from a sample participant pool?
- (d) How are preferences elicited from users for the interface customization domain? What type of elicitation procedure can gauge the user's preferences realistically, even if they are unfamiliar with the system? Since eliciting the user's full utility function is too tedious and error-prone, is there an efficient mechanism that can be used for the procedure? How do these results, elicited offline (i.e., prior to application use), help with online inference about the users?
- (e) How should the system's model of user preferences be designed so that it can keep up with the user's evolving preferences over time?

1.2 List of Contributions

This thesis proposes a framework for developing intelligent software adaptation systems with consideration of the user model in a decision-theoretic manner. The general methodology adopted in this work is to develop formal models that are grounded in AI, while using empirical methods grounded in HCI to learn model parameters and to validate the

usability of the constructed model in a prototype system. The publications produced as part of this thesis have focused mostly on developing new user models for simple domains. Overall, the models developed in this thesis are first tested using simulation experiments. Thereafter, the model is situated in an application context and usability experiments with real users are carried out. In some cases, the work focuses specifically on the design of user experiments to collect data to populate model parameters. As such, the contributions of this thesis lie in the intersection of AI and HCI.

The following is a list of publications produced as part of this thesis. The first publication motivated the thesis direction of formally modeling the customization problem in the context of a communication program for users with traumatic brain injury. The second publication presents a general user model in a decision-theoretic framework in the context of a word prediction software, whose aim is to demonstrate the overall value of a decision-theoretic assistance approach. The third and fourth develop a stochastic model for the user’s mental model of a menu-based application, which we argue is crucial to minimizing the disruption induced by adaptive systems. The fifth paper focuses on identifying and formalizing relevant interaction factors that have been studied in HCI, whose purpose is to help design the user’s utility function in intelligent software adaptation systems. The sixth paper proposes a new elicitation technique that elicits the user’s subjective preferences for novel interfaces in the context of an augmented PowerPoint system, so that the numerical sensitivity of individual utility functions can be better understood.

1. [HLM03] B. Hui, S. Liaskos, and J. Mylopoulos. Requirements Analysis for Customizable Software: A Goals-Skills-Preferences Framework. In *Proceedings of Requirements Engineering (RE)*, pages 117–126, 2003.
2. [HB06b] B. Hui and C. Boutilier. Who’s Asking for Help? A Bayesian Approach to Intelligent Assistance. In *Proceedings of Intelligent User Interfaces (IUI)*,

- pages 186–193, 2006.
3. [HB06a] B. Hui and C. Boutilier. Modeling the Disruption to the User’s Mental Model. *Neural Information Processing Systems (NIPS), Workshop on User Adaptive Systems*, presentation only, 2006.
 4. [HPB09] B. Hui, G. Partridge, and C. Boutilier. A Probabilistic Mental Model For Estimating Disruption. In *Proceedings of Intelligent User Interfaces (IUI)*, pages 287–296, 2009.
 5. [HGIB08] B. Hui, S. Gustafson, P. Irani, and C. Boutilier. The Need for an Interaction Cost Model. In *Proceedings of Advances in Visual Interfaces (AVI)*, pages 458–461, 2008.
 6. [HB08] B. Hui and C. Boutilier. Toward Eliciting Experienced Utilities for Interface Customization. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 298–305, 2008.

1.3 Outline of Thesis

The rest of this thesis is organized as follows. Chapter 2 presents the background concepts on probabilistic and decision-theoretic reasoning and surveys the literature in this area as applied to the intelligent software adaptation problem. As one of the major problems in interface customization is the lack of formal approaches to modeling individual differences, we propose a general decision-theoretic framework in Chapter 3 by modeling the problem of intelligent software adaptation as a partially observable Markov decision process. We identify three crucial components which are presented in subsequent chapters: the user characteristics model, the goal model, and the reward model. Chapter 4 proposes a set of user features relevant to intelligent assistance and presents a generic user model for inferring these features online. Data collection experiments are conducted

to learn more realistic model parameters, simulation experiments are conducted to assess the model performance, and usability experiments are done to evaluate the overall system in which the user model is embedded. Chapter 5 identifies useful goal classes for intelligent assistance and presents a domain-specific model of user goals in the context of authoring slides in PowerPoint 2003. Simulation experiments are conducted to assess the value of inferring the user goal in the decision-theoretic framework, and usability experiments are conducted to confirm the utility of the goal model. Chapter 6 outlines a set of interaction factors relevant to interface customization and formalizes them as part of the reward model. The reward model takes the proposed user characteristics from Chapter 4 to capture individual differences. Data collection experiments are performed to learn a more realistic model of the objective component of the reward model, and elicitation experiments are done to elicit subjective values of the reward model using a new, experiential elicitation procedure. Finally, Chapter 7 summarizes the lessons learned by developing the various models and from the various evaluations conducted in this thesis.

Chapter 2

Background

In this chapter, we review background material for several decision-theoretic (DT) modeling techniques that are used in this thesis. The complexity and generalizability unfolds as we progress through the models. With an emphasis on these modeling techniques, we review literature related to intelligent software adaptation systems and summarize the major obstacles in the area.

2.1 Bayesian Networks and Dynamic Bayesian Networks

A *Bayesian network* (BN) [Pea88], or Bayes net, is a graphical model that represents a joint distribution over a set of n random variables $\mathbf{X} = X_1, \dots, X_n$ with their assignment values denoted as x_1, \dots, x_n . Bayes nets have two components: a graphical component and a numerical component. The graphical component is defined using a *directed acyclic graph* that consists of nodes representing each of the random variables in the joint distribution and directed edges representing direct dependencies (i.e., causal relationships) among the random variables. Specifically, if X_2 has an outgoing edge to X_1 , then X_2 is the parent node of X_1 . We denote the parents of X_i as Pa_{X_i} , with their assignment

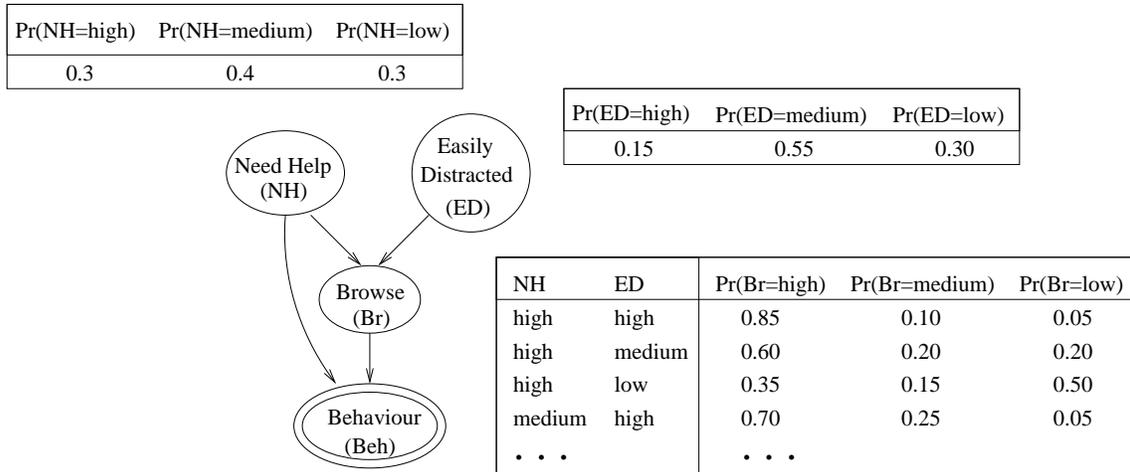


Figure 2.1: Example Bayes net with several CPTs shown. By convention, observations are drawn in double-circles.

values denoted as $pa_{x_1}, \dots, pa_{x_n}$ for the corresponding x_1, \dots, x_n . The numerical component specifies the conditional probability distributions $Pr(X_i | pa_{X_i})$, $1 \leq i \leq n$. The probability of x_1, \dots, x_n is given by:

$$P(X_1 = x_1, \dots, X_n = x_n) = \prod_{i=1}^n Pr(x_i | pa_{x_i}) \quad (2.1)$$

Following this specification, Bayes nets allow us to exploit structural independence by compactly specifying only the local conditional dependencies rather than the full joint distribution.

As an example, a simple Bayes net with four discrete variables is shown in Figure 2.1. This model represents two random variables about a user – whether the user needs help (NH) and how easily distracted the user is (ED). At a high level, this Bayes net models the behaviour of a user at different levels of neediness and distractibility. For example, if the user needs help, then he is likely to display signs of “trouble”, such as pressing undo keys (e.g., backspaces), browsing for more information, or pausing to search for help. On the other hand, if a user is easily distracted, he is likely to show signs of distraction, such as switching between applications and surfing menus without clicking on any menu item.

Structurally, NH has three possible outcomes corresponding to a high, medium, and low levels of neediness respectively. NH does not depend on other variables (i.e., no parents nodes), and influences two other variables (i.e., has two child nodes). The probability of NH having any outcome is roughly uniformly distributed, which is expressed in a table called the *conditional probability table* (CPT). CPTs define the probability distribution of a variable given its parent node values. Now consider the variable *Browse* (Br) which has three outcomes, two parent nodes, and one child node. Its CPT (partially shown in Figure 2.1 only) enumerates the probability distributions over browsing for each combinations of the parents' values. Analogously, the numerical component of a Bayes net involving continuous variables are defined using *conditional probability distributions* instead. The distributions that specify the dynamics involving the network's observation variables are referred to as the *observation model*. In building Bayes nets for user modeling applications, two main areas of focus are the development of a realistic network (structure) and methods for populating the numerical parameters with real data.

Bayes nets encode the *Markov assumption* in that each node is independent of its non-descendants given its parents. For example, *Behaviour* (Beh) is independent of ED given Br . As a result, the joint probability of a Bayes net can be expressed as a product of the local conditional distributions. In Figure 2.1, the joint distribution $Pr(NH, ED, Br, Beh)$ is rewritten as:

$$Pr(NH)Pr(ED)Pr(Br|NH, ED)Pr(Beh|NH, Br) \quad (2.2)$$

Bayes nets can be used to infer the probability distribution of variable values. In the context of customization, our interest is in inferring the state of certain user variables, such as the user's neediness level or the user's current goal. At any point in time, the observed user behaviour can be used to update our belief of those variables. For example, upon observing the user surfing menus, we may want to know what is the probability that the user needs help, i.e., computing $Pr(NH|Beh = surf)$. Alternatively, upon observing the user pausing, we may want to know what is the joint probability that the

user needs help and is easily distracted, i.e., $Pr(NH, ED|Beh = pause)$. This description summarizes the inference task, where observations are recorded as evidence in the network and the variables of interest are then queried.

A general exact inference algorithm is the *clique tree* algorithm [LS88, HD96], also called *junction tree* or *join tree* algorithm. Given a Bayes net, this algorithm first creates a clique tree for the inference operation by moralizing and triangulating the graph, identifying cliques, and connecting all the cliques based on specific criteria. The numerical component in the original Bayes net is represented as potentials in each clique so that inference is done by multiplying and summing over potentials in the clique tree. The clique tree is initialized with the Bayes net's CPTs. To enter observed evidence, a clique containing the observation variable is selected and its observed value is multiplied into the clique's potential. To ensure consistency, message passing is carried out across the tree in two passes. To compute a marginal, a clique containing the variable of interest is identified and all other variables in that clique are summed out. Normalization is performed to ensure the resulting probability is valid. An iterative algorithm (e.g., variable elimination [Dec99]) can also be used for exact inference.

To model dependencies across time, Bayes nets are extended to *dynamic* Bayes nets (DBNs) [DK89]. We focus on a two-stage DBN specified by a set of n random variables \mathbf{U} over two time stages t and $t + 1$. Generally, these variables will involve hidden state variables \mathbf{X}_t , and observable variables \mathbf{Y}_t , where $\mathbf{X}, \mathbf{Y} \subset \mathbf{U}$. In particular, this DBN is a *first order Markov model* such that the variables in time $t + 1$ are directly influenced by variables in stage t , and are independent of the variables in previous stages, i.e., $Pr(\mathbf{U}_{t+1}|\mathbf{U}_{1:t}) = Pr(\mathbf{U}_{t+1}|\mathbf{U}_t)$. Given a Bayes net, a two-stage DBN is built by creating two copies of the Bayes net and adding additional temporal dependencies from the variables in stage t to those in stage $t + 1$. Figure 2.2 shows a simple two-stage DBN that extends the model from Figure 2.1. Here, we see the hidden state variables NH , ED , and Br have temporal dependencies across the two stages, indicating that

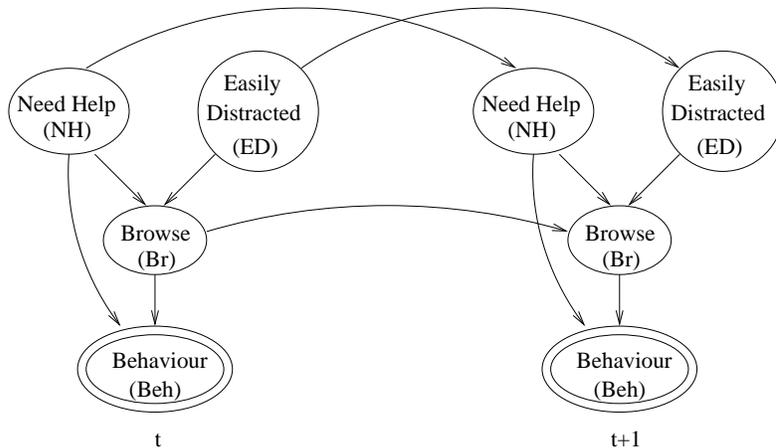


Figure 2.2: A two-stage DBN extended from Figure 2.1.

a user state from one time step influencing the state at the next time step. In other words, this model has three transition functions: $Pr(NH_{t+1}|NH_t)$, $Pr(ED_{t+1}|ED_t)$, and $Pr(Br_{t+1}|Br_t, NH_{t+1}, ED_{t+1})$. For example, the transition function for NH may capture the dynamics that a user who is highly needy now is more likely to sustain that neediness level rather than switching to being not needy at all. In this example, we assume that time stages are defined over an interval of time that is long enough to allow observations such as surfing menus and pauses to be detected.

The numerical component of a DBN are specified as conditional probability distributions. In particular, there is a prior distribution $Pr(\mathbf{U}_0)$, a state transition function $Pr(\mathbf{X}_{t+1}|\mathbf{X}_t)$, and an observation function $Pr(\mathbf{Y}_t|\mathbf{X}_t)$.

The goal of inference is to infer $\mathbf{X}_{1:t}$ given the observed evidence $\mathbf{Y}_{1:t}$. In our example, we are interested in predicting a hidden state (e.g., the user's neediness level) given a history of observations, i.e., $Pr(\mathbf{X}_{t+1}|\mathbf{y}_{1:t})$, where $\mathbf{y}_{1:t}$ is an instantiation of the observed evidence representing a particular history. Clique tree inference as described above can be used for DBNs. At $t = 0$, the clique tree is created and the numerical parameters are initialized with the DBN prior distribution. Over time, a stream of evidence $\mathbf{y}_1, \mathbf{y}_2, \dots$ is observed. At $t = 1$, the system's belief is $Pr(\mathbf{X}_1)$. When \mathbf{y}_1 is observed, evidence

is observed and entered into the clique tree, and the updated belief $Pr(\mathbf{X}_1|\mathbf{y}_1)$ is computed, which is used to compute the marginal of interest for the prediction task, i.e., $Pr(\mathbf{X}_2|\mathbf{y}_1)$. At $t = 2$, the *rollup* step takes place where the old belief is discarded and a new clique tree is initialized with the DBN prior distribution, and the belief $Pr(\mathbf{X}_2|\mathbf{y}_1)$ is incorporated into this tree. Using Bayes rule, this computation can be rewritten as: $Pr(\mathbf{X}_2|\mathbf{y}_1) \propto Pr(\mathbf{y}_1|\mathbf{X}_2)Pr(\mathbf{X}_2)$. When \mathbf{y}_2 is observed, it is entered into the clique tree, and the updated belief $Pr(\mathbf{X}_2|\mathbf{y}_{1:2})$ is computed. Using Bayes rule, this computation can be rewritten as: $Pr(\mathbf{X}_2|\mathbf{y}_{1:2}) \propto Pr(\mathbf{y}_2|\mathbf{X}_2)Pr(\mathbf{X}_2|\mathbf{y}_1)$. This process continues as new evidence is observed over time.

The performance of clique tree inference depends on the size of the cliques that are created, which in turn depends on the independence structure in the DBN. When exact inference can no longer meet the online demands in user interaction scenarios, approximate algorithms are needed. A general approximate inference algorithm is the *Boyen-Koller* algorithm [BK98]. This algorithm enforces independence over parts of the Bayes net or DBN by introducing *clusters* over groups of variables. Specifically, to approximate the joint belief distribution, the algorithm instead computes the marginals over these chosen clusters of variables. Then, it uses an exact inference procedure, such as the clique tree algorithm, to perform inference. The algorithm enforces independence of these variable clusters at each time step by projecting the joint distribution at a time step into the marginals over these clusters. This ensures the required computations remain feasible and bounded, even for large networks.

2.2 Markov Decision Processes

A Markov decision process (MDP) [Put94] models an agent’s long-term decision problem under uncertainty. This agent acts in a world of observable states whose dynamics over time are specified by a stochastic transition model. An action taken in a given state has

consequences on the agent's transition, what future actions should be taken thereafter, whether a goal is reachable, and the rewards subsequently collected. This *sequential* nature is inherent in any realistic decision making problem and can be captured in the way the agent chooses its actions to solve the problem.

States are associated with real-valued rewards (or costs). As an agent transitions from one state to another, the agent accumulates the associated reward. The objective of this agent is to maximize the expected sum of discounted rewards received over an infinite horizon. In particular, this criterion assumes that a reward received n time steps in the future is discounted by γ^n , where $0 \leq \gamma < 1$ is the discount factor. In this setting, the agent's goal is to choose actions that maximizes the expected, discounted future reward. Formally, an MDP is a tuple consisting of $\langle S, A, T, R, \gamma \rangle$ where:

- S is a finite set of states
- A is a finite set of actions
- $T : S \times A \times S \rightarrow [0, 1]$, is a probabilistic transition function, where $T(s, a, s') = Pr(s'|s, a)$, $\forall s, s' \in S$ and $\forall a \in A$, denotes the probability of moving to state s' if action a is taken in state s . Note that $\sum_{s' \in S} Pr(s'|a, s) = 1 \forall s, a$.
- $R : S \times A \rightarrow \mathbb{R}$ is a bounded, real-valued reward function
- γ , the discount factor

To illustrate, we consider an intelligent software adaptation agent who has the ability to suggest help to the user via a pop-up box. Suggesting help in this way could result in assisting a user who needs help, which could lead to positive rewards; however, the same help also risks distracting the user and resulting in a less preferred state that could otherwise be avoided. In particular, there is a cost to every agent action (e.g., interrupting the user), and a reward to being in a certain state. Assuming one could observe the user's neediness and distractibility levels, Figure 2.3 shows the MDP for this example.

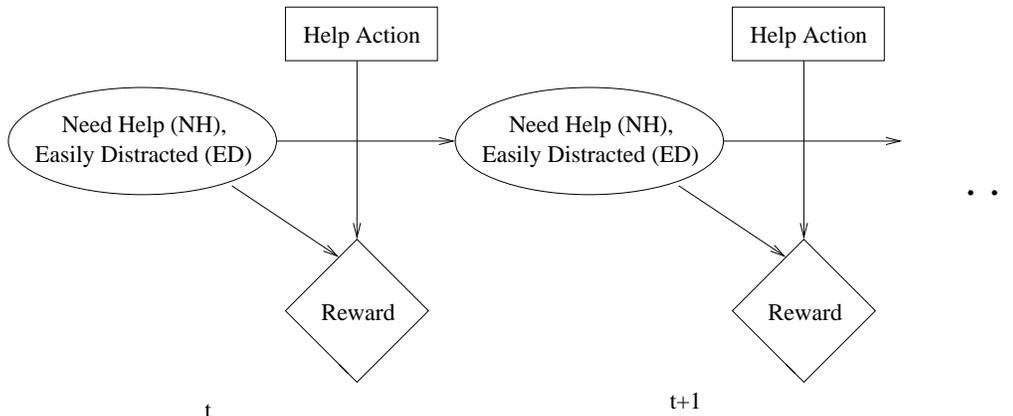


Figure 2.3: Example MDP with two time stages shown. By convention, actions are drawn in a square and rewards are drawn in a diamond.

Note that the state variables in this MDP have been simplified to exclude Br and Beh . The reason is that NH and ED are now assumed to be observed, so there is no longer any need to infer their values indirectly via an observation model or additional hidden variables. In this MDP example, NH and ED are aggregated together so that the transition function models the joint probability distribution of $Pr(NH_{t+1}, ED_{t+1} | NH_t, ED_t)$. In contrast, recall the transition functions in the corresponding DBN example from Figure 2.2 that models the transition dynamics of NH and ED independently. Note that *factored* representation [BDH99] can be used to model MDPs more naturally and concisely.

After the model is specified, the MDP can be solved to find an optimal policy that maximizes the MDP's objective $E_\pi[\sum_{t=0}^{\infty} \gamma^t r^t | s_0 = s]$, where π is the policy being executed, r^t is the reward received at time t , and s_0 is the initial starting state. Based on this criterion, the value function of a policy π is defined as:

$$V^\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r^t | s_0 = s\right] \quad (2.3)$$

The optimal policy for this MDP model is known to be stationary [Put94]. Thus, we represent an MDP *policy* as a mapping from states to actions, i.e., $\pi : S \rightarrow A$, so that

an agent is given a prescribed action at every state. The optimal policy, denoted as π^* , is the policy that maximizes the value function for all the states. The optimal value function is defined as:

$$V^*(s) = \max_a \{R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s')\} \quad (2.4)$$

Dynamic programming can be used to compute the optimal value function in an iterative manner. Initially, starting at stage 0, we set $V_0^\pi = R(s)$. Then, the algorithm iteratively computes the i -stage-to-go value function for a given policy π as:

$$V_i^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_{i-1}^\pi(s') \quad (2.5)$$

Based on Equation (2.5), the Bellman's principle of optimality [Bel57] relates the optimal value function at stage i to the optimal value function at stage $i - 1$ by:

$$V_i^*(s) = \max_{a \in A} \{R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_{i-1}^*(s')\} \quad (2.6)$$

A general solution algorithm is called *value iteration*, which implements Equation (2.6). In particular, at stage 0, we set $V_0(s) = R(s)$. At stage i , we compute $V_i(s)$ by backing it up with $V_{i-1}(s)$ according to Equation (2.6). The terminal condition $\|V_i^\pi(s) - V_{i+1}^\pi(s)\|_\infty < \epsilon(1 - \gamma)/2\gamma$ guarantees ϵ -optimality for the policy derived in the i^{th} iteration, which ensures the policy derived from V_i satisfies $\|V_i(s) - V^*(s)\|_\infty < \epsilon$ [Put94].

An alternative solution algorithm is *policy iteration* in which algorithm directly computes a policy for every state. The algorithm begins by picking an arbitrary policy, π' , and repeatedly improving it as follows:

1. let $\pi = \pi'$
2. compute $V^\pi(s)$
3. improve by $\pi'(s) = \arg \max_{a \in A} \{R(s) + \gamma \sum_{s' \in S} T(s, a, s') V^\pi(s')\}$
4. repeat until no improvement results

The optimal policy, π^* , is obtained by maximizing the value function at each state s . Since the definition of a value function is dependent on a policy, the optimal value function, V^* , is the value function corresponding to the optimal policy. For larger decision problems, abstract representations and approximate solution techniques are available [HSAHB99, SAHB00, BDH99, Gor99].

2.3 Influence Diagrams and Partially Observable Markov Decision Processes

MDPs model decision problems assuming states are fully observable. This assumption is often unrealistic in practice, as not all random variables of interest are observable (e.g., the user variables in the example in Figure 2.3). Thus, a more general approach is to model the *partial observability* of the decision problem. One way to do this is to use influence diagrams. An influence diagram models decision problems by extending a Bayes net with decision nodes that represent the agent's actions and utility nodes that represent the value of the actions. Figure 2.4 shows an influence diagram extended from our Bayes net model in Figure 2.1. In this figure, a new decision node that defines the help actions available to the agent is introduced. The utility of the decision is a function of the help action and the user state (as defined by NH and ED). This influence diagram resembles a one-stage MDP with the exception that NH , ED , and Br are hidden variables (i.e., cannot be observed directly).

While influence diagrams are typically used to model problems involving multiple decisions over a finite horizon [SB10], we restrict our attention to problems with one decision node only in our customization examples and applications discussed in this thesis. To evaluate influence diagrams with a single decision node, a simple procedure with the following steps can be used [RN95]:

1. enter the evidence into the appropriate nodes

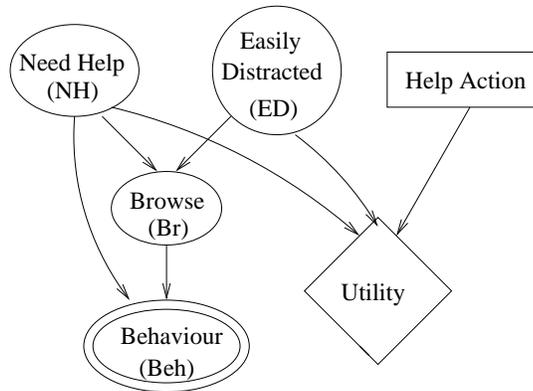


Figure 2.4: Example influence diagram extended from Figure 2.1. By convention, decision nodes are drawn in boxes and utilities are drawn in diamonds.

2. for each agent action:
 - (a) set the decision variable to that action
 - (b) using a standard probabilistic inference algorithm, calculate the posterior probabilities for the parent nodes of the utility node
 - (c) calculate the utility of the action
3. choose the action with the highest utility

Given the observed evidence, this procedure selects the agent’s action according to the principle of maximum expected utility (MEU):

$$A^* = \arg \max_A Pr(S)U(S, A) \quad (2.7)$$

where A^* is the optimal help action, and S is the state defined by NH and ED in the example from Figure 2.4. Other solution methods that compute policies, that map observed evidence into choices of one or more actions or sequences of actions, are discussed in [SB10].

A more general approach to model partially observability of a decision problem is to use a partially observable Markov decision process (POMDP). In essence, a POMDP

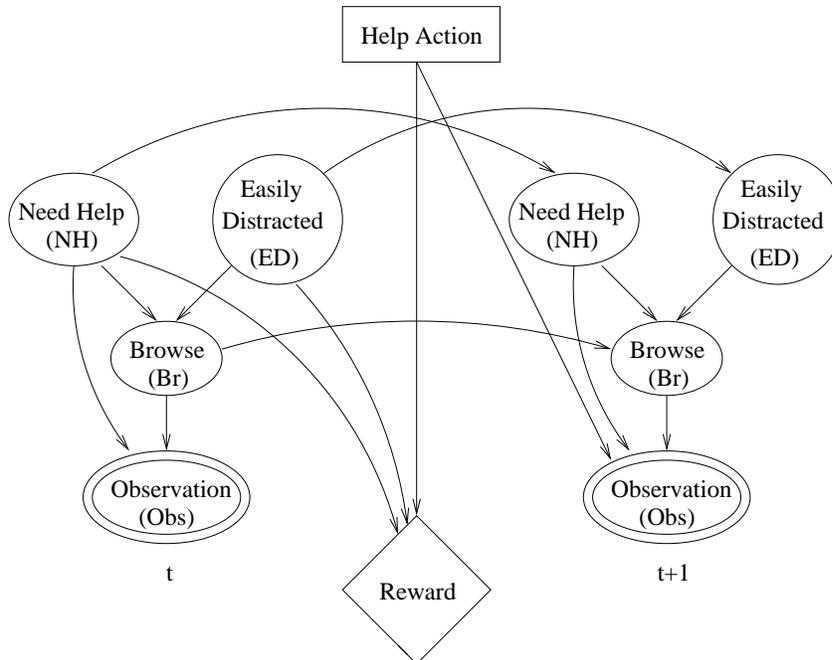


Figure 2.5: Example POMDP with two time stages shown, extended from Figure 2.2.

extends the MDP with a set of observations and a stochastic observation model. Formally, a POMDP is defined as a tuple $\langle S, A, T, R, O, Z, \gamma \rangle$ such that:

- S, A, T, R, γ as defined in an MDP
- O , set of observations
- $Z : S \times A \times O \rightarrow [0, 1]$, observation function to express the probability of experiencing o' after executing a to get to state s' , typically rewritten as $Z(s', a, o') = Pr(o'|s', a), \forall s' \in S, \forall o' \in O, \forall a \in A$

Our intelligent software adaptation problems are modeled as a POMDP in order to provide a general framework. However, we do not solve a POMDP in this thesis, as approximation techniques are used instead. As such, we define the POMDP model and discuss the relevant approximation techniques used in this thesis below. A high level description of POMDP solution methods is provided for reference.

Figure 2.5 shows an example of the POMDP model extended from the DBN in Figure 2.2 by including the system action and reward function. In contrast to the *Beh* observable variable used in the DBN, here, the new variable *Obs* includes behavioural user actions as defined in *Beh*, as well as other kinds of observations that the user may exhibit in response to the help action, such as considering help, accepting help, and rejecting help. In contrast to the MDP model, since the state is not fully observed, this POMDP must maintain a belief distribution over the user variables *NH*, *ED*, and *Br* and make decisions under uncertainty about the user. In general, the system maintains a belief distribution over the state, $bel : S \rightarrow [0, 1]$ such that $\sum_s bel(s) = 1$. In effect, $bel(s)$ indicates the probability that s is the true state.

The process for updating the system’s belief distribution is referred to as *belief state monitoring*. At the start of the system interaction, the belief distribution is set to an initial belief state, bel_0 at $t = 0$. At each time step, the agent experiences an observation and takes an action, both of which are used to update bel as defined in Equation (2.8):

$$bel_t(s') = \alpha \sum_{s \in S} bel_{t-1}(s) T(s, a_{t-1}, s') Z(s', a_{t-1}, o') \quad (2.8)$$

where α denotes the normalizing constant. Belief state monitoring often imposes a severe online computational burden if the state space is large. However, belief state monitoring can often be made tractable if the system dynamics and observation model can be represented concisely using, say, a DBN [BK98]. Such representations exploit independence in the factored model and solve the problem without enumerating the state space.

The objective of the POMDP is to find a *policy* that maximizes the expected discounted sum of rewards. Given the unobservability of the state space, one way to represent a POMDP policy is to map the belief distribution over states to actions, i.e., $\pi : bel \rightarrow A$. Just as in the MDP setting, a value function is used to evaluate a policy. Given an initial belief state bel_0 , the value of executing a policy π starting at bel_0 is defined as the expected total discounted rewards, i.e., $V^\pi(bel) = E_\pi[\sum_{t=0}^{\infty} \gamma^t r^t | bel_0 = bel]$, where π is the policy being evaluated with the belief state bel . Using this objective, we

can order policies such that π_i is better than π_j if $V^{\pi_i}(bel) \geq V^{\pi_j}(bel)$, for all belief states bel . Analogous to the equation for MDP in Equation (2.6), the Bellman equation for POMDPs is defined with respect to belief states as follows:

$$V^*(bel) = \max_{a \in A} \{R(bel, a) + \gamma \sum_{bel' \in Bel} T(bel, a, bel') V^*(bel')\} \quad (2.9)$$

where Bel denotes the set of all belief states. Finally, the optimal policy, π^* , is obtained as in Equation (2.10):

$$\pi^* : V^*(bel) \geq V^\pi(bel), \forall \pi, \forall bel \quad (2.10)$$

Analogous to the MDP setting, value iteration and policy iteration are two classes of solutions used to find the optimal policy for POMDPs [Son71, Mon82, Che88, Han98, MPKK99, PB04]. However, due to the added ability to model planning problems under partial observability, computing exact solutions for POMDPs is an intractable problem in the general case [MHC99]. To this end, various approximation methods have been explored, including point-based estimation of value functions (e.g., [PGT03, SV04]), stochastic finite-state controllers (e.g., [Han08]), and belief state compression (e.g., [RG03, PB03]). Moreover, the use of belief state monitoring coupled with a myopic policy, such as taking the action with maximum expected utility, is a myopic approximation to solving the POMDP. This particular approach is equivalent to the simple procedure described earlier for evaluating influence diagrams with a single decision.

2.4 Applications for Intelligent Software Adaptation

The design of an intelligent system typically involves a model of the agent's environment and the decision making component [RN95]. In the case of applications for intelligent software adaptation, the agent's environment includes a model of the user and the application state. Ongoing system-user interaction allows the system to observe the user and update the user model to make appropriate decisions over time.

Under this view, user actions are inputs to the intelligent system and system decisions are outputs. Directly learning the mapping from inputs to outputs to build a user model would follow the *discriminative* approach, where the outputs are designed to maximize some objectively quantified loss function. As an example, the SUPPLE system renders an interface that minimizes the combined costs of navigating and manipulating widgets based on a history of user traces [GW04]. For example, the lights in a classroom can be turned on/off or dimmed to a certain level, and different widgets (e.g., slider, check boxes, radio buttons) can be used to operate the lighting function. A branch-and-bound constrained search algorithm is used to find an optimal interface rendering that results in complete searches between 0.5 second to 6 minutes under various simulation conditions (e.g., type of constraint propagation and variable ordering), while the best rendering was typically found at a fraction of the time. Given the same functional requirements, SUPPLE demonstrates the ability to generate different interface configurations under different device constraints (e.g., screen size) and user traces (e.g., frequent use of a subset of functions).

Another interface customization example is the FolderPredictor that suggests folder shortcuts for the user's current task in the context of a larger system called TaskTracer [BHD06, SBD⁺05]. The FolderPredictor has access to a set of user-defined, manually labeled tasks so that it can associate folders and usage statistics to each task. By default, when the user wants to open a file in the Windows XP operating system, a dialog box appears showing a list of files from a default folder along with five shortcuts (e.g., "My Computer" and "Desktop") displayed on the side of the dialog box. Among these shortcuts, three have been permanently replaced by FolderPredictor recommendations that jointly minimizes the cost of mouse clicks needed to access the desired folder (given the existing folder hierarchy), in expectation of a discounted recency distribution. Based on longitudinal data from four users, FolderPredictor suggestions reduce clicking cost by 50% on average in comparison to the default folder mechanism, even though the default

method predicts the correct folder more often. The cost/effort of recovery — selecting the wrong folder and getting back to the right one — was not reported for either method.

Discriminative approaches to learning user interests have been used in other domains as well, such as information retrieval systems that implicitly learn user interests based on search keywords [STZ05], collaborative filtering and recommendation systems that map ratings of similar users to system recommendations (e.g., [RIS⁺94, SBH02]). These methods infer user interests, and therefore, user preferences, directly from usage patterns, and have been shown to work well empirically. In contrast, *generative* approaches in user modeling attempt to develop explanatory models that model user features explicitly in order to account for the observed user behaviour. These approaches allow for a principled way for encoding prior knowledge that can be used to constrain the large, unknown preference space of users. From a design and development perspective, generative models help us understand factors that influence behaviour and changes in preferences. The choice between a discriminative and a generative approach is subject to ongoing debate which we do not pursue here. For reasons of increased interpretability and support for incorporating prior knowledge, we adopt the generative approach in our work and focus our literature review below strictly on model-based approaches.

Traditionally, user modeling research focused on modeling user goals, with some effort being devoted to modeling the user’s knowledge, abilities, and preferences in the context of those goals [Kob01]. In such cases, the system’s objective is to minimize the effort exerted by the user while carrying out these goals. Early attempts in modeling user characteristics were motivated by the need to create systems that adapt to the needs of users with communication and interaction difficulties [Kob95]. More recent developments have placed some attention toward maximizing user satisfaction by modeling the user state (e.g., neediness as a function of goal difficulty [HBH⁺98], learning style [ZC03], attitude toward the system [CM05, KBP07], or cognitive resources impacted by system actions [Jam01]). In addition to modeling the application context, the utility of an intelligent

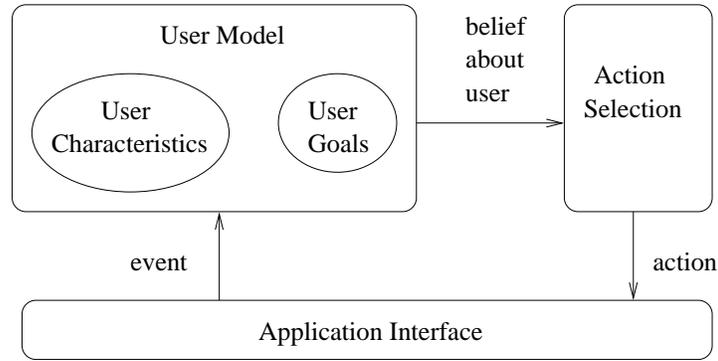


Figure 2.6: Generic system components in applications for intelligent software adaptation.

system is not only a function of its ability to predict user goals accurately, but also a function of its ability to maximize the ease of the interaction as perceived by the user. Putting this together, we propose the generic architecture illustrated in Figure 2.6. In particular, we highlight two components of the user model — user characteristics and user goals. Depending on the specific implementation, these two components may have interdependencies between them. The output of the user model is the system’s belief about the user. This information is used by the Action Selection module to choose intelligent actions according to certain criteria. Using this generic architecture to guide our discussion, we review literature that models both components of the user in a decision-theoretic framework for tailoring the functionality, presentation, navigation, or interactivity level of software applications.

The Lumière project by Horvitz et al. is perhaps the first attempt at developing a decision-theoretic application that models both user characteristics and user goals [HBH⁺98]. In the context of software applications, the authors proposed an influence diagram for providing intelligent help based on (uncertain) system beliefs about the user’s skills, goals, and the task history shown in Figure 2.7(a). Multiple Bayesian networks were built: some focused on context-specific assistance, while others offered were general assistance in the application. The nature of automated assistance is designed to give

users advice on how tasks are best carried out in the application. The focus of the work is on the development of an extensive DBN that infers user goals based on application states shown in Figure 2.7(b). An intermediary event language was developed to bridge the gap between system events and observation patterns of interest in the model. Two user characteristics are modeled: skills and neediness. Persistent user competency (e.g., skills) in the application can be set by the user (e.g., via widget input) and can also be inferred by the successful completions of a set of *indicator tasks*. Finally, the model infers whether the user needs help currently, and compares this probability to a user-specified threshold (“volume control”) to decide whether help should be offered. A “Wizard of Oz” experiment was conducted to better understand when users need help and how users respond to different kinds of help. In particular, human advisors (acting as “wizards” behind the scene) observed and gave advice to users under the perception that the advice was delivered by an intelligent agent. The qualitative feedback guided the design of the model observations. It is reported that the model was constructed using expert knowledge. A prototype of the model was created for Microsoft Excel. In addition, a shallower model that reasons about thousands of goals, but with no user skills, no user neediness, and limited event mapping was included into the Microsoft Office Assistant. Model assessment was mentioned but not reported.

To get a better understanding of the threshold policy used in the Lumière project, we turn to the action selection component of a related project called LookOut [Hor99a]. In this application, the system processes email content and infers the user goal of whether the user wants to schedule an appointment in the calendar or not (G). This system does not model user characteristics. Possible help actions are: to do nothing ($\neg A$), to invoke the calendar, thus, allowing the user to schedule an event (A), or to engage in a clarification dialog (D). Knowing the utility of offering help and doing nothing in advance enables the system to precompute a threshold for its policy as illustrated in Figure 2.8(a). In this analysis, the upper plane corresponds to the action with the highest utility for

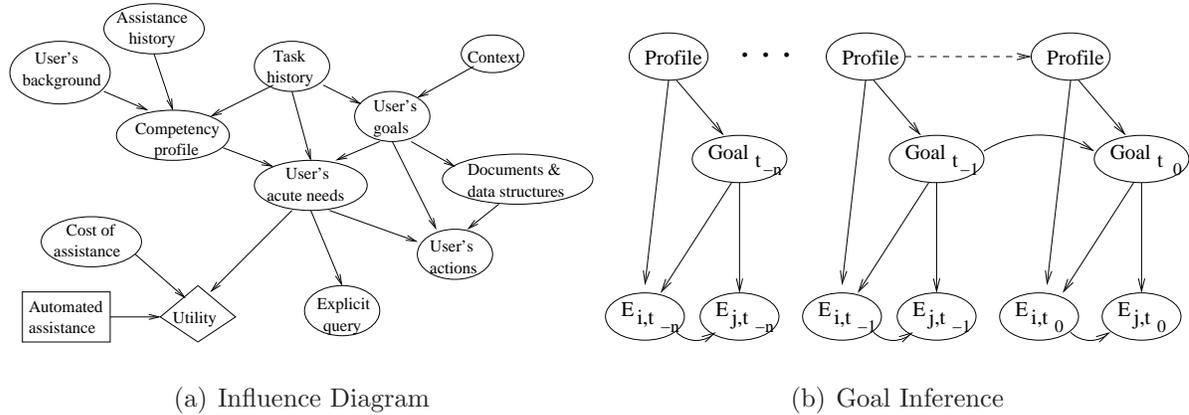


Figure 2.7: Models from Lumière for (a) deciding to provide user assistance and (b) inferring the current user goal (taken from [HBH⁺98]).

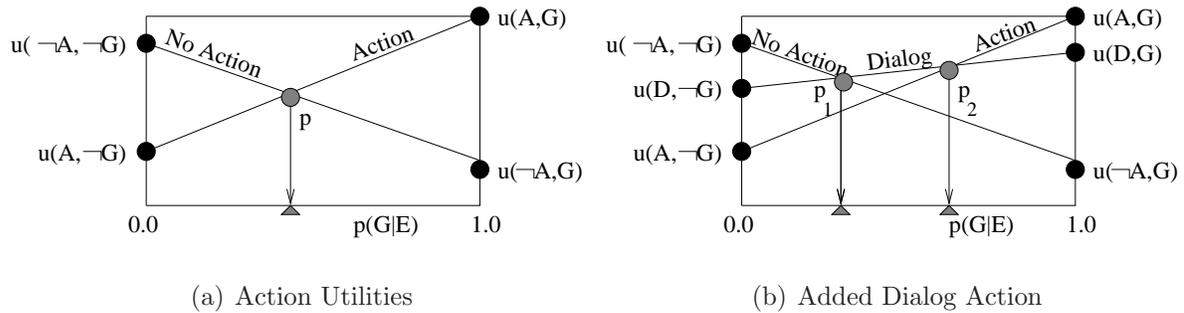


Figure 2.8: Graphical analysis for (a) expected utility of action versus inaction, and (b) with consideration of an additional dialog clarification action (taken from [Hor99a]). The upper plane corresponds to the action with the highest utility for that region.

that region. From this graphical analysis, the system policy is to invoke the calendar if the inferred $Pr(G) > p$ and do nothing otherwise. Figure 2.8(b) shows the resulting analysis of adding a dialog action, where the system policy is to engage in dialog if $p_1 < Pr(G) \leq p_2$, to invoke the calendar if $Pr(G) > p_2$, and to do nothing otherwise. Using a threshold policy simplifies the computational requirements in the action selection module. Note that different combinations of utilities result in different thresholds. The user may manually override default utilities and thresholds by specifying values through the application interface.

Subsequent work by Horvitz et al. continues the development of decision-theoretic

assistance for software applications, with emphasis on issues such as interruption cost, user's attentional state, the value of probing the user to label his current user state, and deployment in other domains (e.g., [HA03, HJH99, KH05, HP00, AH02, HKS⁺05]). This body of literature provides a general decision-theoretic framework for intelligent assistance, complimented by extensive probabilistic models that infer various user goals and characteristics, demonstrating that probabilistic inference is highly tractable in these domains. Assistance utility is restricted to the consideration of the benefits of saving the user from carrying out tasks manually, and the cost of interrupting the user given a variety of system actions.

Next, we turn to the literature in the area of intelligent tutoring systems. In this context, the intelligent system is a tutor who maintains a student model of the user by observing the student work on problems (i.e., goals) and tailors its help actions accordingly. For example, DT Tutor uses a dynamic decision network to model the interaction between the student and the tutor [MV00]. This system models the student's characteristics (specifically, morale and independence), knowledge of the domain, and progress on the domain problem. Tutor actions include taking an action on a specific problem topic, with possible actions such as prompt, hint, teach, positive/negative feedback, do a step of the problem, and do nothing. Utility of the action depends on the student's knowledge and progress on the problem, morale, independence, and the specific tutor action. Details of the model, its parameters, and decision making are not reported. Simulation experiments report several exact and approximate inference algorithms took between 8 to 108 seconds to choose a tutor action. Descriptive system behaviour suggested its ability to choose actions that support the student and mimic human tutor reasoning, e.g., to foster student independence, to prevent student failures, and to teach domain knowledge rules rather than problem-specific steps. This project demonstrated the benefits of a decision-theoretic approach for intelligent tutoring systems and its ability to model multiple objectives and mimic tutor behaviour.

Table 2.1: Methodology for designing decision-theoretic intelligent tutoring systems (taken from [MM01]).

1	Randomized data collection
2	Model generation
3	Decision-theoretic strategy implementation
4	On-line adaptation
5	Evaluation

CAPIT is a tutor that teaches basic capitalization and punctuation rules of English to young children [MM01]. The central contribution of this work is the methodological steps outlined in developing and evaluating intelligent tutoring systems, summarized in Table 2.1. The proposed methodology is *data-centric*, focusing on data collection (Step 1) to automatically learn a predictive model (Step 2) that incorporates online learning techniques (Step 4). CAPIT was used to demonstrate this methodology. A Bayes net is learned from data representing the domain problems and the outcomes of each solution attempt. Although student characteristics are not explicitly modeled, the Bayes net include a proxy that keeps track of the number of errors made by the student. Thus, all the variables are observable. Pedagogical guidelines from learning theories are encoded into the system’s utility function in the action selection module so that the tutor chooses the action that maximizes teaching objectives under uncertainty. These actions involve selecting appropriate problems and error messages for the student. Unlike much of the previous work, the model structure and parameters for the Bayes net are learned from data to maximize the predictive performance of the student model. Several learned models varying in levels of complexity were compared, and some statistical significant differences were found. A post analysis by the authors showed that there was much regularity in the data, explaining the similar predictive performance of these learned models. Model parameters are adapted dynamically to individual students by using

Dirichlet priors with a moving window of incoming observations that discount older data. A simple simulation of a “good” and a “bad” student was conducted to illustrate the system’s behaviour toward the two types of students. A field experiment conducted with three classes of students was carried out. Class A is a control group who did not use the system at all. Class B used the system equipped with a random policy for selecting problems as well as feedback messages. Class C used the decision-theoretic tutoring system as described. Overall, students in Class C showed the most improvement between pre and post test results. Students in Class C also exhibited behaviour that indicated they were more engaged and learned at a faster rate than those in Class B.

Prime Climb is an educational game designed to help young children improve their factorization skills through a series of climbing exercises [Con02]. This intelligent tutoring system uses a dynamic decision network to decide on the kind of help to provide when mistakes are observed. The main contribution of this work is the development of a probabilistic model of learning goals (e.g., have fun, win, watch the player fall), user traits (e.g., extroversion, agreeableness), domain knowledge, and user affect variables (e.g., shame, joy) founded in cognitive theory of emotion. The design of the tutor’s utility function is not documented, although it is reported that the system takes chooses actions with maximum expected utility. Details for learning the model structure and parameters are reported elsewhere [ZC03, CM09a]. Extensive evaluation on the accuracy of the predictive model was carried out based on data collected from students using Prime Climb in classrooms. However, the overall utility of the system, such as its role in supporting the student’s learning progress, was not reported. Subsequent work by Conati et al. continues the development of affect models in the context of decision-theoretic tutoring systems, with emphasis on issues such as data-driven model refinement, use of eye tracking, and impact of model accuracy on student performance (e.g., [ZC03, CM09a, CM07, CM09b]). This work has also inspired other work on probabilistic models of affect in a decision-theoretic setting [PMI05, LZZ⁺06].

The literature reviewed for intelligent tutoring systems emphasized the importance of the predictive accuracy of the user model, with secondary focus on the decision making aspects of the system. In contrast to the myopic decision policies reviewed in these projects, Jameson et al. proposes intelligent systems that consider the long term consequences on users [JGHM⁺01]. As an example, this work models the problem of delivering multiple instructions separately or together in one bundle. This model was first constructed as a Bayes net, which models the user’s distractibility designed to approximate the available resources in the user’s working memory. The model was extended to an influence diagram, where the system’s utility function is a weighted combination of the user’s execution time and error rate from carrying out given instructions, defined independently of the system action and the user’s distractibility. Finally, the model was converted to an MDP where the system decides to give one instruction or wait for the user to execute the instruction(s) already delivered over a finite horizon. In this MDP, the user’s cognitive resources are assumed to be fully observable — an assumption that should be relaxed using a POMDP model instead. Data collection experiments were conducted to learn the numerical parameters of the Bayes net model from users. By inspection, system policies tailored its actions toward the user’s distractibility as well as different weights given to the two factors in the utility function. Using different parameter settings, a comparison of the resulting policies from various models concluded that actions prescribed by more general policies are not easy to predict and not easy to reproduce with simpler models. In other words, handcrafting heuristic rules will likely not mimic general, principled behaviour. Subsequent work by Jameson et al. continues to develop predictive models of user resources using physiological and speech features, to apply MDPs in mobile applications, and to explore mechanisms that make decision-theoretic applications more usable in realistic settings (e.g., [SJ04, JKM⁺09, BJ01, BJJA05]).

Recent work from HCI extended a dual-interface version of MS Word [MBB02] to allow users to manually switch from the default interface showing the full set of function-

ality to a personalized interface showing only a subset of functionality based on system recommendations [BCM07]. In contrast to the literature reviewed above, this work focuses on the assessment of the impact of system recommended adaptations. When the user decides to modify the personalized interface, the system suggests an additional function for inclusion based on the savings gained in its selection time if it were added to the personal interface than if it remained in the full interface, versus the cost in the selection time of other functions due to this added function in the personal interface. A discussion was provided to explain the need for modeling user expertise and expected usages in the system. A user study was conducted to compare the dual-interface system with and without customization suggestions (adaptive and adaptable, respectively). For each system, participants carried out two tasks that required users to follow step-by-step instructions on using various editing functions while creating a document. While these tasks required a large number of menu selections and repetition in function usage, the precise tasks to carry out, functions used, and frequency distributions were not provided. A total of 12 participants completed the study, 8 of whom customized the interface in both adaptive and adaptable conditions. While differences in task completion times were marginally significantly faster in the adaptive condition, the time taken to customize the interface was significantly faster in the adaptive case than if participants added functions manually. Among all the system suggestions, 96% of them were accepted by participants. No functions were deleted in any of the cases — follow-up discussions indicated that the design of the tasks and/or the duration of the study did not provide the context or motivation for users to remove unused functionality.

2.5 Challenges

Based on the literature reviewed above, we summarize four challenges in advancing the development of decision-theoretic assistants for customized software interaction.

2.5.1 Designing the Utility Function

In order for an intelligent software adaptation assistant to help the user, it must be able to model the factors that determine the quality of the interface and interaction that is being tailored. Recall the methodology proposed in Table 2.1 that suggests the tutoring system’s strategy should be designed in alignment with pedagogical guidelines so that the tutor can choose actions that best support the student’s learning process. CAPIT accomplished this by encoding two learning theories directly into the system’s utility function. In the same way, software adaptation assistants need to have the appropriate design criteria encoded in the utility function so that the intelligent system can act in concordance with pre-established design and interaction principles. The MDP example from Jameson et al. [JGHM⁺01] addresses this by modeling execution time and error rate as components of a weighted utility function. Here, minimizing execution time is an objective that competes with the objective of minimizing error rate, so the tradeoff between the two is expressed by their respective weights. The ability to represent the necessary tradeoffs in the utility function enables the intelligent system to evaluate the impact of its own actions and select the one that is most suited to the design criteria.

Such factors are often application-specific. For example, a system may be designed to minimize interface “bloat” by considering the number of functions used versus the number of functions displayed [McG00]. Many more examples of such factors have been suggested in the HCI literature, such as: processing time, time savings, efficiency, physical demand, ease of use, satisfaction, confusion, mental demand, discoverability, predictability [War01, GCTW06, CGG07, BGBG95]. As evident in these examples, terminology used to describe such factors are often ambiguous and inconsistent across research groups. Furthermore, these factors typically evolve out of a post-questionnaire self-reporting procedure given to users for feedback. Thus, the majority of them do not have formal models that are readily available to be used in defining an intelligent system’s utility function. For these reasons, a more general approach is needed to help integrate existing HCI prin-

principles into the system's action selection module so that multiple, competing objectives can be modeled systematically and appropriate tradeoffs can be made in a principled manner.

2.5.2 Subjective Utilities: Modeling and Elicitation

In addition to designing the factors that are relevant in the system's utility function, the problem of modeling individual differences and eliciting the numerical values of the function arises. For example, an action that pops up a dialog box on the screen may offer some beneficial information to a user who needs help but at the same time interrupts the user's current work flow. In other words, the utility of system actions needs to be defined with respect to the user's current state. i.e., $U(A, S)$, where S may represent various user characteristics such as interruptability level, distraction level, neediness level, and so forth. Modeling the utility function in this way enables the system to indirectly infer the user's preferences over system actions as a function of (or conditional on) the current user state. Thus, as the user state changes, the corresponding conditional preferences change also. From this perspective, the key to modeling user preferences now lies with modeling the relevant interaction factors and the associated user characteristics that contribute to the overall utility function. As discussed in the literature review, there has been little attention focused on the design of utility functions in customization applications.

Given the structure of a utility function, the numerical values need to be identified. In particular, since the software adaptation assistant is designed to help the user, the system's utility function need to represent the user's subjective utility function. For example, using execution time as a measure (as in the case of Jameson et al. [JGHM⁺01]) gives objective values because task execution times are calculated based on system events. From a usage perspective, some users may have the same preference for different durations of execution times simply because they cannot distinguish the changes in the durations. The ability to identify these perceptions in a quantitative and systematic way across

users is needed as part of designing the system’s utility function. As mentioned above, Horvitz et al. investigated a model of interruption cost [HA03]. First, a data collection experiment is set-up to ask users to express (label) their interruptability level (along with many other observable variables such as time of day) and their willingness to pay for a system action (interruption) to go away. The collected data was then used to create a Bayes net to infer the user’s interruption cost at run-time. Similar approaches may be used to tackle other interaction benefits and costs. In addition, advances in preference elicitation can also be applied in this domain [CGNS98, CKP00, CKO01, Bou02].

2.5.3 Partial Observability Coupled with Long Term Policies

As discussed above, the importance of partial observability and long term policies are both needed in developing intelligent software adaptation systems. To our knowledge, the combination of these two aspects have not been used in this domain. As presented earlier, POMDPs are a general way to account for both of these aspects. Recent applications of POMDP models have been developed in areas such as robotic assistants for the elderly [PMP⁺03], providing hand washing guidance for Alzheimer’s patients [BPH⁺05], dialogue management [WY05], office navigation [SV04], and grasping with a robotic arm [HKLP07]. The feasibility of the methods used in these domains as applied to the customization domain needs to be explored.

2.5.4 Continual Learning of Model Parameters

Learning an empirical user model is typically done by conducting (offline) experiments that collect labeled data by associating observations to user states (user goals or characteristics). When the amount of data is not enough, or when the user’s behavioural model needs to be adapted over time, online methods for collecting this data are needed. In the case of the CAPIT system [MM01], continual learning was achieved by modeling its parameters using Dirichlet priors and updating them with incoming observations.

This approach works in their case because their Bayes net did not involve any hidden variables. However, if hidden user characteristics are involved (rather than using observable variables as proxies), then alternate methods are needed for online learning (such as probabilistic inference that adapts to user-specific observations).

Horvitz et al. explored the value of querying the user about the hidden state in a message alerting system for precisely this purpose [KH05]. At a high level, the value of a query is formulated as the difference between the expected gain of the system with intelligent alerting behaviour over default behaviour when it has the knowledge of the newly labeled data compared to the expected gain without such added knowledge, subject to the query cost over a given horizon. Changing the horizon results in different querying behaviour — the system tends to be reluctant to ask queries with a short horizon and tends to ask a lot of queries with a large horizon. Evaluation using labeled data from two users showed that the system asked few queries, positive utility gain overall, and significant accuracy improvement in comparison to alternative query strategies that is random and that is based on minimizing uncertainty. The use of active learning strategies designed to select queries in a decision-theoretic way can also be explored further in the customization domain.

Chapter 3

A DT Framework for Intelligent Customization

As we reviewed in Chapter 2, the development of intelligent assistants has benefited from the adoption of DT approaches that enable an agent to reason about and account for the uncertain nature of user behaviour in a complex software domain. To summarize, these approaches typically try to help the user accomplish a task more efficiently and easily by estimating user-specific information, such as the user's current task, whether the user needs help, or how frustrated the user is with the system. In addition, complex tradeoffs must be assessed when deciding if and when to offer help to a user, hide a specific function, etc. For example, deciding to offer help must balance the uncertain assessment as to whether help is needed, the costs of unwanted interruption, the benefits of providing the right type of help, and the costs of providing the wrong type of help or of doing nothing when help is needed. As discussed in Chapter 2, DT models have been demonstrated in a variety of domains (e.g., [HBH⁺98, Con02, MM01, HA03, BJ01, JGHM⁺01, BPH⁺05]), allowing a system to make the right decision based on such decision-theoretic tradeoffs.

In the same spirit as [MM01], we propose a methodological framework for developing an intelligent software adaptation system in this thesis. Our goal is to provide devel-

opment guidelines that help researchers and programmers design, learn, and evaluate such systems. In our case studies, we limit our attention to general desktop software with standard keyboard and mouse devices only, and do not include observations from special devices (e.g., speech, video, physiological inputs). However, our framework is generalizable to more specialized applications.

To demonstrate our framework, we will focus the majority of our efforts on the development of the user characteristics component of the user model and the Action Selection component as outlined in Figure 2.6. The research efforts presented in this thesis will highlight how we address the challenges of designing and modeling an individual’s utility function for intelligent software adaptation systems. As our framework adopts the decision-theoretic paradigm, we will make use of the formal modeling tools reviewed in Chapter 2. As such, we begin by formalizing the interaction dynamics in Section 3.1, then present our framework in Section 3.2, and finally relate the focus back to the research objectives posed from Chapter 1 in Section 3.3.

3.1 Developing the Decision-Theoretic Model

We view the intelligent software adaptation problem as a planning problem under uncertainty. In comparison to traditional planning techniques, a DT approach is one that considers the uncertainty in the model (e.g., noise in the observations, stochastic effects of actions) as well as the utility that reflects the consequences of actions. Following existing approaches, we adopt a DT framework to design an agent that makes rational decisions about its actions under uncertainty. We propose to model the interaction process between the system and the user as a *sequential* stochastic process where the user moves from state to state. Here, *states* reflect the user’s attitudes and abilities (cognitive, motor, etc.) as well as the user’s goals and system’s application context. By viewing relevant user information as part of the stochastic environment and monitoring it over time, the

agent is able to tradeoff interaction costs and benefits induced by a potential adaptation action with respect to its belief about the (generally evolving) user state. Under this perspective, we use a POMDP to model the system’s underlying intelligent software adaptation problem, while the actual solution mechanism employs some approximations (e.g., the use of a myopic policy). We first take a closer look at the variables at play in an interaction session between the user and the system.

3.1.1 Interaction Scenario

Consider a user authoring slides in the PowerPoint application as a concrete example. A typical scenario involves the user entering information (e.g., by typing text or creating graphics objects, such as diagrams, tables, and pictures) and editing it until the user is satisfied with the presentation. At a closer look, this authoring process involves a lot of repeated actions and sequences of actions throughout the creation of a presentation (and potentially across multiple presentations). This repetition includes highlighting new terminology using particular combinations of font styles, emphasizing the slide’s punchline using certain indentation settings, or connecting and grouping duplicated nodes and arcs in the same way when drawing complex diagrams. The consistency exhibited in these repetitive actions provides opportunities for an intelligent system to observe and learn user-specific patterns. We refer to these repetitive user actions as *user goals*, so that the agent can learn them and help the user with them by doing auto-completions, creating shortcuts or one-click icons on the interface, and suggesting them in a toolbar for ease of access. Keeping example help actions and user goals in this application domain in mind, we formalize the interaction between a user and a help system below.

We model software interaction as a series of *episodes* where we assume, for simplicity, that the user attempts to complete a single goal in each episode. In a given episode, the user has a certain goal in mind (unknown to the system) and carries out a sequence of application-specific actions to achieve it. When that goal is completed, the user con-

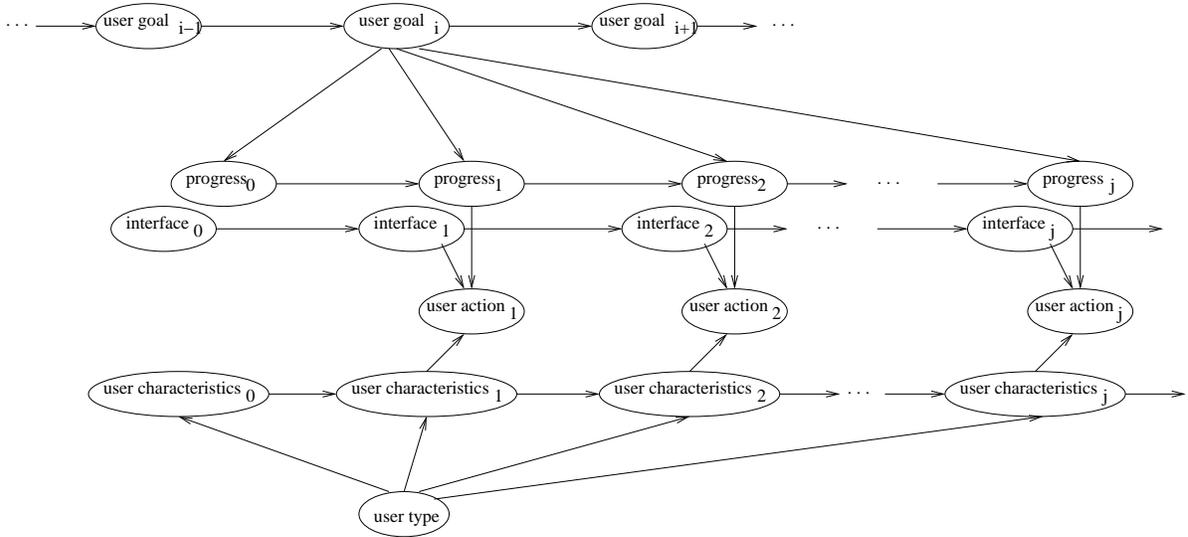


Figure 3.1: Episodic interaction, omitting the influence of system help on the progress and interface states. The episode of the i^{th} goal has time steps $t = 0..j$, where the j^{th} interface state and user characteristics for episode i influence the 0^{th} interface state and user characteristics, respectively, in episode $i + 1$.

tinues onto the next episode to pursue a new goal. This interaction repeats. During the interaction, the system estimates the user goal by monitoring observable *progress* (i.e., in terms of the history of user actions relative to the application state that designates goal completion) and the *interface* state (i.e., the kind of information presented in the interface, whether automated help is available, etc.). Figure 3.1 illustrates the sequential interaction, with the i^{th} episode expanded to illustrate the variables and their dynamics. In this case, the i^{th} episode consists of j user actions. We assume that the user has both *static* and *transient* variables that dictate behaviour. We refer to the static variables as the *user type* because their values do not change over the course of the interaction. On the other hand, we refer to the transient variables as *user characteristics*, since their values tend to change more frequently during the course of an episode.

This interaction model presented in Figure 3.1 is akin to that proposed interaction model from the Lumière project (which was reproduced in Figure 2.7(b)). Our model

differs in two ways. First, we eliminate the direct influence of the user profile on the user goal, and only keep the direct dependency that the user profile has on the way that the user goals are carried out. In this way, our approach assumes that user goals are derived from a larger problem solving context that is independent of the user’s personality or attitudes. Unlike systems that consider different classes of users that have different rights of access and tasks in a system [Vas04, HBH⁺98], our model focuses on general goals that are applicable for different kinds of users. Second, we separate the user profile into user type and user characteristics in order to distinguish static features of the user (that do not change over the course of an interaction session with the system) from transient features of the user (that change frequently during the interaction).

Figure 3.1 does not model interaction with the system. To account for an intelligent system in this interaction model, we add adaptation actions and utilities to each time step. We do not illustrate the extended diagram here, as the model will be reformulated as a POMDP in the next section. Here, we describe the additional dynamics required to model the system interaction.

At a given point in time, the system can decide to help the user complete the desired goal by offering assistance or rearranging the application interface in a way that makes the subsequent interaction easier for the user to achieve the goal. (Alternatively, the system may also decide it is best to not interrupt the user and do nothing.) From the system’s perspective, taking an action may influence the progress state — e.g., when auto-completion is accepted by the user, which results in transitioning from the progress state to the goal state. The system action may also influence the interface state — e.g., when a toolbar suggestion presents a set of shortcut macros on the screen, which allows the user to consider, accept, or reject the shortcuts suggested. System actions do not directly influence the user state; user states evolve as a result of the changes in the application state due to the system’s actions. The system can only learn about the user — goals, type, characteristics — by observing his behaviour (or asking a query and

observing the response to the query). For example, the user may dislike the interruption or the way in which the help was offered and show behaviour that suggests annoyance and frustration. On the other hand, a user may welcome system suggestions because they help him progress further in achieving his goals, and show this attitude by always accepting help even when it does not match the desired goal perfectly. The information about how the system should act is encoded in the system’s utility function so that it models the factors that direct the system’s reasoning and behaviour — e.g., the cost of interruption, the suggestion mechanisms that annoyed the user, the savings that resulted from an accepted suggestion. Therefore, the system action and the user state directly impact the utilities. These dynamics are incorporated in the POMDP in the next section.

3.1.2 POMDP-DAISI

This section defines a POMDP model for Developing Intelligent Software Interaction and Assistance (letters rearranged as DAISI, pronounced like “daisy”). The dynamics and independence assumptions built into this model is based on the episodic interaction scenario presented in the previous section. We define the assistant POMDP in this episodic environment using the following factored representation:

- State space $S = Y \times C \times G \times P \times I$, where Y, C, G are variables representing the user and P, I are variables representing the application state. In particular:
 - Y is the set of static variables describing the user type (e.g., general tendency to be frustrated, general tendency to be independent). Note that these variables are static across episodes.
 - C is the set of transient variables describing user characteristics (e.g., current level of frustration, current level of neediness).
 - G is the set of static variables describing user goals (e.g., typing a word, highlighting a phrase to be bold and italics). In contrast to user type variables,

these goal variables are static within episodes only.

- P is the set of variables describing the user’s progress toward goals in the application state (e.g., the user having highlighted the selected phrase to be bold, but not yet italics).
- I is the set of variables describing the interface in the application state relevant to predicting user goals or user characteristics (e.g., toolbar suggestion is presently on the screen, a slider widget for configuring the level of adaptive help is set to low).
- Action space A is the set of adaptation actions available to the assistant (e.g., presenting a toolbar with 5 icons of a certain quality).
- Transition model $T(S_{t-1}, A_{t-1}, S_t) = Pr(S_t|S_{t-1}, A_{t-1})$ is a probabilistic function that maps states at time $t - 1$ and system actions to a distribution of states at time t . We define a factored transition model to express the independence of the model dynamics across the different types of state variables as discussed previously in Section 3.1.1. Specifically, we have:
 - User type and user goal are static within the episodic model and their transition dynamics are independent of A , so we have $Pr(Y)$ and $Pr(G)$ respectively.
 - Transition dynamics for user characteristics are also independent of A , so that $T(C_{t-1}, A_{t-1}, Y, C_t) = Pr(C_t|Y, C_{t-1})$.
 - Transition dynamics for application state variables are deterministic, so that $T(P_{t-1}, A_{t-1}, G, P_t) = Pr(P_t|G, P_{t-1}, A_{t-1})$ and $T(I_{t-1}, A_{t-1}, I_t) = Pr(I_t|I_{t-1}, A_{t-1})$ are deterministic.
- Observation space O is defined as the set of user actions. We consider two types of observable user actions making up this space such that $O = O^C \cup O^G$, where O^C is a set of behavioural observations indicative of user characteristics (e.g., jamming into

the keyboard indicates frustration), and O^G is a set of observations indicative of the user working towards a goal (e.g., clicking on the bold icon indicates a highlighting goal).

- Observation model $Z(S_t, O_t) = Pr(O_t|S_t)$ is a probabilistic function that maps states to a distribution of observations. Following the graphical dependencies depicted in Figure 3.1, we assume the observation model is independent of the user type given user characteristics and independent of the user goal given the progress state, i.e., $Pr(O_t|C_t, I_t, P_t, Y, G) = Pr(O_t|C_t, I_t, P_t)$ in the factored representation. These are reasonable independence assumptions because one may design progress states to capture the necessary information about how the user goal influences observations, and design user characteristics to capture the necessary information about how the user type influences observations. Using this representation, one may wish to decompose the observation models into a simpler form such as $Pr(O_t^G|P_t, I_t)$ and $Pr(O_t^C|C_t, I_t)$ in order to alleviate the computational requirements involved.
- Reward model $R(S_{t-1}, A_{t-1})$ is a mapping from states and actions to reals, expressing the (immediate) reward of executing A_{t-1} with respect to the current state S_{t-1} . Generally speaking, the system’s immediate reward consists of the benefits gained by carrying out A_{t-1} minus the cost induced by it. Since different users may respond to the same adaptation in different ways, we model the *objective* benefits and costs of A_{t-1} , as well as the *subjective* value of that action as perceived by the user. These components are specified as follows:
 - *benefits*($C_{t-1}, P_{t-1}, A_{t-1}$) is a function that captures the perceived benefits of A_{t-1} toward achieving the current goal. As a result of the specific action taken by the system, A_{t-1} , the system measures an objective change in the “distance” between the progress state, P_{t-1} , and the (ultimate) goal state. This distance is then expressed as the *perceived value* with respect to the

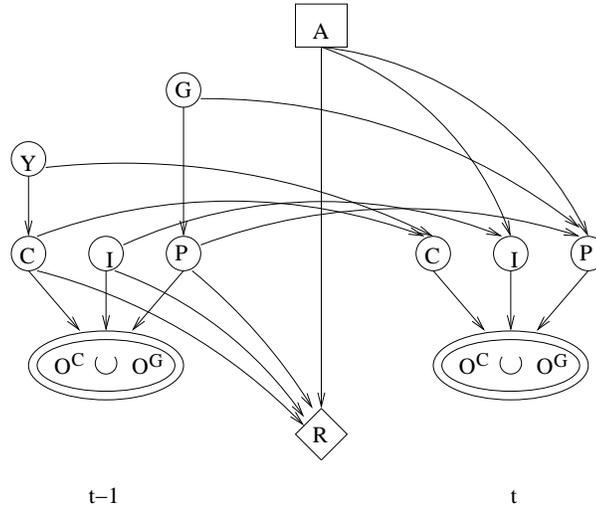


Figure 3.2: The graphical representation of POMDP-DAISI.

specific user characteristics, C_{t-1} .

– $costs(C_{t-1}, I_{t-1}, A_{t-1})$ is a function that defines the perceived cost of A_{t-1} .

The system measures the *annoyance* of A_{t-1} in the context of the interface state, I_{t-1} , in an objective way. This cost is then expressed as the perceived value with respect to the specific user characteristics, C_{t-1} .

Altogether, the reward model is: $R(C_{t-1}, I_{t-1}, P_{t-1}, A_{t-1}) = benefits(C_{t-1}, P_{t-1}, A_{t-1}) + costs(C_{t-1}, I_{t-1}, A_{t-1})$.

Figure 3.2 illustrates the POMDP-DAISI model. In general, Y may directly influence Z and R . In order to simplify the structural dependencies, one may define user characteristics to capture user types so that the observation and reward models are independent of Y , as shown in this figure.

In comparison to other DT approaches, the DAISI model encompasses more user information. Aside from the literature reviewed in Chapter 2, we are unaware of other decision-theoretic approaches that model both user goals and user characteristics. In effect, other work employs a simpler state space. This simplification leads to a simpler observation space, observation model, transition model, and reward model altogether. As

well, our literature review reveals that when user characteristics are modeled, observable variables are often employed as a proxy for the underlying user features. As a result, the logistics and design needed to probe the hidden user features is eliminated and online computational requirements for inference is reduced. Thus, this simplification leads to simpler learning and inference tasks.

3.1.3 Cross-Episodic Interaction

POMDP-DAISI models the dynamics in an episode of interaction, where we assume the user is only interested in one goal in each episode. As illustrated in Figure 3.1, across episodes, the system needs to maintain changes in the interface state as well as its belief about the user. The interface state can simply be updated by setting the last interface state of episode i to be the first interface state of episode $i + 1$. Maintaining user information across episodes is more involved, as we explain below.

Within an episode, user characteristics are updated at each time step. At the start of an episode, the system has a prior distribution over the user characteristics. Given an observation, the system computes its posterior distribution. This repeats at each time step until the episode is over. As the system’s beliefs about the user characteristics are kept up-to-date throughout the episode, they can be used directly as the prior distribution at the start of the next episode. The inference steps for user goals within an episode are quite similar to those for user characteristics. Across episodes, the prior distribution of user goals is updated based on observed goal completions from the previous episodes. Since we assume the user pursues a new goal in the next episode, no other information is carried over. This simplifying assumption can be relaxed by modeling additional goal information. For example, if there were correlations between goals across episodes (e.g., ordering of goals), we could capture it in $Pr(G^i|G^{i-1})$.

The aforementioned procedure reflects the scenario where the system has a reasonable model about general users in the population and that information is applicable to the

individual user at hand. In other words, online learning is not needed to adapt the model parameters. Such an assumption may be reasonable in terms of the distribution of user types in a population, or how user types influence user characteristics, or even the observation model used to infer user characteristics. However, this assumption is often less realistic in modeling user goals. As an example, consider the PowerPoint highlighting goals such as changing a phrase to bold and red to emphasize its importance. Although we may encode general population behaviour within the system to express users' tendencies to highlight important phrases when authoring slides, the font pattern used by one user may be very different from the choices of another. Thus, the system needs to learn the specific goals that are relevant to individual users. For this reason, the system would ideally learn a personalized goal model that has a user-specific goal observation model and goal distribution. These two pieces need to be updated upon the completion of a goal (i.e., at the end of an episode) so that a more accurate goal model can be used in inferring user goals in subsequent episodes.

3.1.4 Solution Strategy

By solving the POMDP, we obtain a policy for intelligent software adaptation assistance that maps belief states to actions. In other words, by updating the system's belief about the user and using it with the policy, we obtain the optimal adaptation action with respect to the specific belief distribution. In practice, the software adaptation POMDP developed for particular interaction domains may be too large to solve. Throughout this thesis, we simplify the POMDP model and develop MDP and DBN models instead. When an MDP model is used, an optimal policy with an infinite horizon is obtained, under the assumption that the relevant user states are fully observable. In the case of a DBN, it is used in conjunction with the POMDP reward model to obtain a myopic MEU policy that identifies the action with the maximum expected utility. While the use of myopic policies relieve much of the computational requirements, they are unable to

take exploratory actions that may not yield immediate benefits even if those actions may have considerable long term gains. The inability to take actions that probe at the value of information is the biggest tradeoff in using myopic solutions.

3.2 The DAISI Framework

In this section, we elaborate on the methodology that makes the implementation of POMDP-DAISI more transparent. As a start, the developer must have in mind the target application, a set of user goals in this application, and a set of system actions that are designed to help the user. Based on this information and any other development constraints (e.g., computational requirements), the developer will need to identify the relevant interaction factors associated with the system actions in the application. (We present in detail how these choices are made in Chapter 6.) This forms the basis of the reward model to be used, as well as identifying the relevant user characteristics involved in determining the subjective utility of system actions. Once these input parameters have been determined, the POMDP-DAISI model needs to be developed.

Given the POMDP-DAISI model structure as defined in Figure 3.2, the next task is to define the numerical parameters of these models. Recall that some of these parameters are deterministic or defined as identity functions. The remaining parameters that need to be defined are:

- Prior (conditional) distributions: $Pr(G_0)$, $Pr(Y)$, $Pr(C_0|Y)$
- Transition model: $Pr(C_t|Y, C_{t-1})$
- Observation model: $Pr(O_t|C_t, I_t, P_t)$
- Reward model: $R(C_{t-1}, I_{t-1}, P_{t-1}, A_{t-1})$

These parameters can be learned empirically, handcrafted based on expert knowledge (e.g., previous literature results), elicited from users, or a combination of the these.

As mentioned earlier, the goal model should be adapted to individual users. Thus, $Pr(G_0)$ needs to be empirically updated online. Also, note that the observation model is application-specific, as it involves user actions defined in terms of specific system and application events. Finally, a solution strategy needs to be chosen and implemented accordingly.

Once the model is completed, the remaining step is evaluation. We do not attempt to discuss evaluation methodologies here. Rather, we highlight two general approaches for evaluation:

- Simulation experiments
- User studies

We view these two approaches as complimentary to each other. Although simulation experiments are not often used in HCI literature, they are a great way of gaining insights into overall performance. Relative to user studies that involve logistics and recruitment efforts, simulations are much more economical and efficient. Moreover, in models that hypothesize many user types, simulations can be used to exhaustively evaluate system performance against all the user types in the model, while user studies can be used to focus on a subset of cases. In an evaluation, measures that should be considered include:

- Computational Feasibility
- Predictive Accuracy
- Overall Utility

The majority of the evaluations reviewed in Section 2.4 focused on assessing the predictive accuracy of the user model. While accuracy is an important metric, a system that has a poor decision strategy, even if the predictive accuracy of its model is high, will likely result in user dissatisfaction. The use of simulation tests is an excellent way to evaluate predictive accuracy as well as computational feasibility (e.g., speed of inference

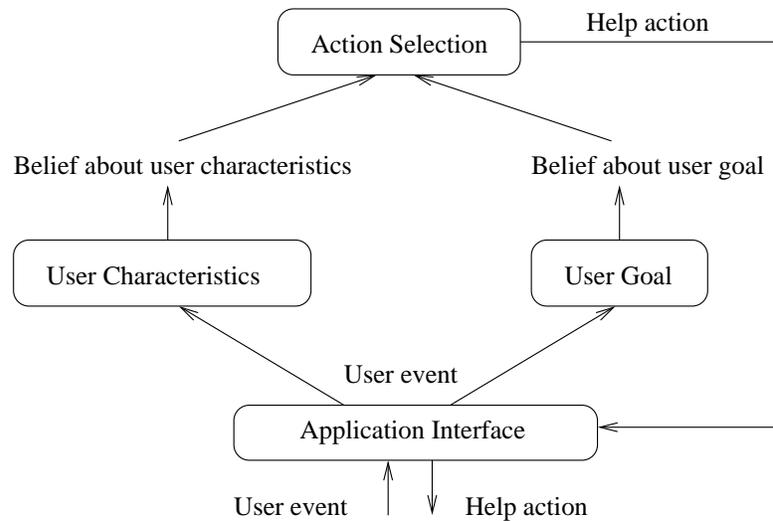


Figure 3.3: System architecture for DAISI.

procedures). Moreover, simulations provide a systematic and controlled environment for developers to assess the potential value of the system under various assumptions. This approach allows developers to fine tune the system parameters before having users evaluate its overall utility. Examples of utility metrics include a comparison of execution time and percentages of automated help accepted by users. The case studies included in subsequent chapters will provide more examples for specific applications and models evaluated.

Figure 3.3 shows the system architecture we use to implement POMDP-DAISI. In comparison to the general architecture presented at the beginning of the literature review in Chapter 2, this one decouples the user characteristics model and the user goal model into separate components for simplicity.

Overall, there are three components of interest. The User Characteristics Component and the User Goal Component implement a model for inferring user characteristics and user goals, respectively. The Action Selection Component implements the mechanism used to select the best action for the user. In the case of an MEU policy, this component computes the expected utility of each system action. Thus, this situation would require

the reward model as well. In the case of a MDP or POMDP policy where the policy is simply a mapping, this component simply returns the prescribed system action. In either case, the reward model is used either to compute the best course of action in real time or to solve for a policy offline.

3.3 Relevance to Research Objectives

This section describes how the design of DAISI addresses the specific research questions raised in Section 1.1. To recapitulate, the research questions in general concern the issues of (i) modeling individual differences and (ii) modeling system actions and consequences in a decision-theoretic framework for the intelligent software adaptation problem.

To model individual differences, we use state variables Y to capture different types of users and state variables C to represent evolving user characteristics and attitudes toward the automated system. These variables are used to develop an explanatory user model that relates user types and characteristics to observable behaviour, O . In software design where profiling users and identifying user groups are used as common heuristics to customize software, our user types can also be utilized for these purposes. The concept of user types (which we use interchangeably with *user groups* and *user profiles*) do not change during the course of a software interaction session, so we model them as static variables. On the other hand, some user features change frequently and have an impact on how the user behaves at the computer and how they perceive adaptive help. For example, consider a user type dimension that models one’s “tendency to be frustrated” and a user characteristic that captures one’s current “level of frustration”. A user who has an extremely high tendency of frustration may result in high frustration levels more often than not. However, we would expect a user who has an extremely low tendency of frustration to not display high levels of frustration at all. Over time, as the characteristics and attitudes change, so does the observed interaction behaviour.

Our approach to modeling individual differences is to focus on learning such user characteristics efficiently in real-time. From a practical perspective, our work has been prototyped in several testbed applications, including a text editor with word prediction help, a predictive drop-down menu interface, and a PowerPoint toolbar help. We explore the variation of individual differences and investigate online computational requirements in these settings.

The second research question we address in this thesis is how system actions and their consequences are modeled. As specified in the DAISI reward model, user characteristics are defined as parameters and used to explain how they influence one's perception of the costs and benefits of automated help. In this way, the agent can compute the utility of its actions in expectation, with respect to its belief about the user, and use this information to evaluate whether the action is beneficial to the user before executing it. When multiple system actions can be suggested simultaneously (e.g., multiple macro icons can be suggested together via a single toolbar), the system can compute a *joint* expected utility of the set of actions. In this way, the reward model plays a key role in deriving an optimal policy for interacting with users.

One major aspect in developing the reward model is to formalize a set of interaction factors prominent in HCI. These factors are used to model the various benefits and costs associated with particular types of system actions in a help system. Benefits are defined in terms of the system action and the degree to which help improves the current state (e.g., the time required by the user to move from the current cursor location to select a suggested macro icon, which results in completing the desired goal without executing the separate events manually). This quantification provides an objective measure of the benefits of a system action. Similarly, an action may have costs, such as annoyance to the user in the context of how it is presented on the interface (e.g., the action may induce visual occlusion that blocks the user's view of parts of the screen, or it may interrupt the user's workflow, etc.). Since not everyone perceives interaction the same way, our

formalism separates these objective values from subjective utility. In this view, utility must be elicited from users. We do this by developing a novel elicitation procedure for eliciting interface preferences with users.

Chapter 4

Modeling Individual Differences

This chapter develops the User Characteristics Component in the DAISI framework as described in Chapter 3. We first present empirical evidence that different people have different preferences in perceiving the value of automated customization actions in Section 4.1 and then describe a set of user characteristics that influence the perceived utility of automated suggestions in Section 4.2. Section 4.3 and Section 4.4 document our experience in developing the User Characteristics Component in two separate case studies.

4.1 Do People Have Different Preferences?

A basic premise of the DAISI framework is the assumption that different people have different preferences in the context of intelligent software adaptation. As mentioned in Chapter 1, recent research advances have tackled the need to create customizable software that tailor to individual preferences in different settings (e.g., [HBH⁺98, HLM03, BCM04, GW04]). In this section, we describe the empirical evidence supporting this assumption in the context of the applications of interest in this thesis. The details of the empirical evidence we draw from are presented more fully in a pilot usability experiment in Section 4.3 and a preference elicitation experiment in Section 6.7.

In the first experiment, four users were asked to use a Dvorak keyboard to type in

some printed text into a text editor with word prediction support under various control conditions. Each of these conditions was set-up with a different policy, so that users would experience different kinds of word prediction help from the system. Specifically, one condition is set so the system offers no help at all (i.e., the user simply typed the printed text into the editor), while another condition offers word completion suggestions every time a keystroke is entered by the user. From the user's perspective, these two conditions created a static user interface since the interactivity level of the system remains the same throughout an interaction session — the former policy is to NEVER offer help while the latter is to ALWAYS provide help. In addition, two adaptive policies were employed in the other conditions to let us to explore the value of adaptive system help. These two policies were parameterized by the estimated quality of help associated with potential word predictions/suggestions. In brief, one adaptive policy was designed to provide suggestions based on word prediction accuracy, while the other provided suggestions based on both predictive accuracy as well as the potential objective effort saved if the user accepted help.

Since using the Dvorak keyboard to type was unfamiliar to each of the users, we found that all the users preferred to have word prediction help to not having any help at all. One user commented that even short words such as “and” and “to” should have word predictions. This comment suggests that the user found the typing task more difficult and preferred to use system suggestions more than the other users. In comparing the four policies, one user preferred the two static policies equally over the two adaptive policies. This user has a strong preference for predictable system behaviour. The other three users, however, indicated that more suggestions were better, even if the suggestions did not contain the target word of interest. This preference reflects the users finding the task to be too tedious and/or difficult.

In the second experiment, four preference elicitation procedures were conducted with a total of thirty-eight users. In the experiment, users were presented with an elicitat-

tion query about their interface preferences in the context of an augmented version of PowerPoint 2003, with emphasis on the use of highlighting important words and phrases while authoring a slide presentation. An example highlighting task would require a user to select a phrase and change it to a specific font pattern (e.g., red, bold, italics, and shadowed). To assist users with highlighting tasks, the application records previously executed highlighting patterns as macros and suggests them as icons in a toolbar. Users were asked to compare the effort needed to carry out such highlighting tasks manually with the effort required to use system suggestions. In both cases, users were asked to also consider any potential effort involved in correcting mistakes made in the process. The specific utility function under investigation in this experiment involved three parameters: *neediness* — a hidden user characteristic variable which indicates the level of neediness of the user (e.g., based on the difficulty of the task and application environment); *length* — the number of icons displayed in the suggestion toolbar when system help is presented; *quality* — the amount of savings relative to the manual effort of executing the highlighting task if the user were to accept the best matching icon appearing in the suggestion toolbar. In the elicitation procedure, we defined these parameters as discrete variables and elicited a subset of the values in order to reduce the cognitive requirements of the experiment.

Although the elicited utility functions varied widely across users, two major patterns emerged: (i) the utility functions were monotonically non-decreasing in suggestion quality, with fixed neediness level and suggestion length, and (ii) the utility functions were monotonically non-increasing in suggestion length, with fixed neediness level and suggestion quality. The first pattern indicates that when neediness and length are controlled for, the utility of the system suggestion is the same or higher as quality increases. For some users, suggestion quality of a medium level may be perceived as offering high utility, while other users may perceive the same suggestion quality to be of low utility. Likewise, the second pattern indicates that when neediness and quality are controlled for, the util-

ity of system suggestion is the same or lower as suggestion length increases. For some users, the utility of suggestion toolbars with ten icons was noticeably lower than those with five icons or one icon, while for others, the utility of suggestion toolbars with five icons were just as low as those with ten icons.

Overall, we find that different users have varying preferences even in very simple automated help settings, such as offering word completions while the user is typing or offering toolbar suggestions while the user is highlighting phrases.

4.2 Relevant User Characteristics

As discussed in Section 4.1, different people perceive the same automated help differently. Whether a user accepts automated help depends on certain attributes of the assistance (such as the quality of the help, the length of the suggestions, etc.) as well as user-dependent variables that explain different preferences across users. As motivating examples, we consider the following scenarios: (a) someone who is easily distracted may find automated assistance costly because it prevents the user from completing the task; (b) someone who currently needs help with a difficult task may benefit greatly from partially correct suggestions that helps the user identify the next steps; (c) someone who is not very independent may not mind receiving imperfect suggestions as much as someone who is highly independent; and (d) someone who is frustrated with the system now is likely to become more frustrated with further interruptions and suggestions. Our goal is to model these differences via a set of user characteristics which we will use to parameterize the utility function to explain these varying preferences.

In Section 6.1 below, we use previous literature findings to aggregate a set of interaction factors that are relevant to determining the utility of automated suggestions. For each of these factors, we identify a set of user characteristics which we use to explain individual differences in the perceived style of interaction. The characteristics we identify

are:

- *Frustration* — The user’s frustration level indicates the user’s attitude toward the current computing environment. The user’s frustration level may increase or decrease based on the system’s behaviour. For example, a frustrated user who receives automated suggestions irrelevant to the immediate task may become even more frustrated. The user’s frustration level influences the perceived utility of automated suggestions. In particular, the same automated help may be perceived as having lower utility when the user is frustrated than when the user is not frustrated. Frustration also influences the perceived cost of visual occlusion and interruption of a suggestion in a similar way. For example, more frustrated users will perceive a higher cost (both in terms of occlusion and interruption) for the suggestion than users who are not frustrated.
- *Distractibility* — The user’s distractibility level indicates the user’s general tendency to be distracted while working in a computing environment. A user who is more easily distracted may find automated assistance distracting and have a harder time resuming a task as a consequence of the distraction. Distractibility influences the perceived utility of automated suggestions because easily distracted users may find the help less valuable than users who are not easily distracted. Distractibility also influences the perceived cost of bloat since excessive functionality creates more chances for the user to be unnecessarily distracted.
- *Independence* — The user’s independence level indicates the user’s general tendency to work independently in a computing environment. For example, a user who is highly independent may not consider or accept help even if it could save the user a lot of effort. For this reason, the user’s independence level influences the perceived utility of automated suggestions. In particular, independent users would find the same automated help less valuable than dependent users.

- *Neediness* — The user’s neediness level indicates the user’s attitude toward the immediate task. The user’s neediness level may change based on the difficulty of the task (with respect to the application environment) and the system’s behaviour. For example, a needy user who gets partially correct help from the system may find the assistance highly valuable because it brought the user closer to completing the task. For this reason, the user’s neediness level influences the perceived utility of automated suggestions. Specifically, needy users will generally find the same automated help more valuable than those who are not needy.
- *Speed* — The user’s speed indicates the user’s general cognitive ability to process information on the screen and motor ability to move an input device to a target. While these two abilities may be modeled as two separate variables, we consider them as a single user speed for simplicity. As an example, when presented with a list of words (such as those displayed in a system suggestion using word prediction), different users take different amount of time to search for a target word (or a close match to it) and select it from the list. Note that some users carry out the act of processing and selection in parallel; some users may begin to move the mouse toward a general area before cognitively identifying the desired target. The speed in processing information is a type of user skill that can be improved upon through practice, but it typically does not change (much) during the course of the interaction. Speed influences the cost of processing time of automated suggestions. In particular, users who are slow at processing may find automated help with many suggestions highly dissatisfying even if one of those suggestions is correct.
- *Concentration* — The user’s level of concentration indicates how focused the user is on the current task. For example, a user who is in the middle of an intensive task may be more involved than he is at the beginning or at the end of that task. The user’s level of concentration influences the cost of occlusion and interruption of

automated suggestions. For example, more information that is occluded from the user's visual view has a higher cost for users who are highly concentrated on their tasks than users who are not focused at all. Likewise, the higher the concentration level, the more costly it is to interrupt the user.

- *Tolerance* — The user's tolerance level indicates the user's general ability to tolerate the use of an interface with excessive functionality. For example, some users prefer a personalized interface that displays only a selected subset of functions for navigation convenience, while others prefer an interface that displays everything to increase potential function accessibility. This tolerance level influences the perceived cost of interface bloat, since an interface showing a lot of excessive functionality may be highly preferred for users who have a high tolerance level.
- *Strength of the Mental Model* — The strength of the user's mental model indicates how well the user knows certain aspects of the application. For example, a novice user may learn where certain menu items are located in an application, and over time, build a mental representation of those locations so he would know where to access those menu items in the future. His mental model of the application may change as a result of a system action (e.g., hiding unused menu items, adding shortcuts in a more convenient location). The strength of a user's mental model influences the cost of disruption induced by adaptive systems that change the interface in an effort of helping the user in future interaction. In particular, a user who has learned certain function attributes well (e.g., location of menu items) may find it very disruptive for the system to change those attributes, while a novice user who has little to no knowledge of that information may not find that system action disruptive at all.

These characteristics form the basis of the User Characteristics Component of our DAISI framework which we instantiate in two case studies. In the first case study presented in

Section 4.3, we develop a user model with characteristics relevant to suggestion savings in the context of word prediction help. Specifically, the characteristics we focus on are: frustration, independence, neediness, and distractibility. In detail, we discuss the steps in developing this user model, the computational procedures involved in the intelligent system, the user model’s role in deriving the system policy, and the utility of a system that makes use of such a user model in comparison to other help systems without a user model. In the second case study presented in Section 4.4, we develop a representation of the user’s mental model. The application domain is adaptive menus, where the intelligent system may add or hide menu items over the course of the interaction. In detail, we present our formal model of the user’s mental model of menu item locations, the procedures for updating the mental model based on the system’s actions, the role of the mental model in estimating the cost of disruption of a system action, and the utility of a system that explicitly represents the user’s mental model in comparison to other help systems that do not account for the user’s mental model.

4.3 Characteristics that Influence Savings

This section presents the first case study where we focus on developing the User Characteristics Component in the DAISI framework [HB06b]. The core of this component consists of a user model, which we formalize in Section 4.3.1. Following recent probabilistic techniques applied in the user modeling literature (e.g., [HBH⁺98, AZN99, HA03, CGV02, ZC03]), we model our user with a DBN and propose to explicitly infer the “user type” — a composite of personality and affect variables — in real time. We design the system to reason about the impact of its actions given the user’s current attitudes. To demonstrate the benefits of this approach, Section 4.3.2 describes an application domain used to show the utility of this DBN user model, namely, an adaptive text editor that provides word completion suggestions. In that section, we will also describe the User

Goal Component used to predict the words in the typing task and the system’s decision policy in the Action Selection Component. Through simulation experiments reported in Section 4.3.3, we show that user types can be inferred quickly, and that a myopic policy offers considerable benefit by adapting to both different user types and changing attitudes. Section 4.3.4 develops a more realistic user model, using behavioural data collected from forty-five users to learn model parameters and the topology of our proposed user types. Using the new user model, Section 4.3.5 reports on a usability experiment conducted with four users and four comparison policies. These usability experiments, while preliminary, show encouraging results for our user model and adaptive policy.

4.3.1 A Bayesian User Model

We begin by proposing a user model formalized as a DBN that allows an automated assistant to learn about its user.

4.3.1.1 User State and User Types

First, we consider the factors that influence whether a user accepts help from an automated assistant, and the value of such assistance, as illustrated in Figure 4.1 (left). Whether a user accepts automated help depends on the quality of the assistance (*QUAL*) as well as the user’s tendency to work independently (*TI*) and the amount of attention that is directed toward considering help (*CONS*). For example, a user who is highly independent may not consider or accept help even if it matches the target goal perfectly. The degree to which a user might consider help depends on the user’s current attitudes toward the automated agent and general personality traits while working in a computing setting. Relevant user attitudes include those directed toward the computing environment, such as frustration (*F*), and those toward the immediate task, such as neediness (*N*). Relevant personality traits include the user’s tendency to get distracted (*TD*) and tendency to work independently (*TI*) on a computer. These influences are illustrated in

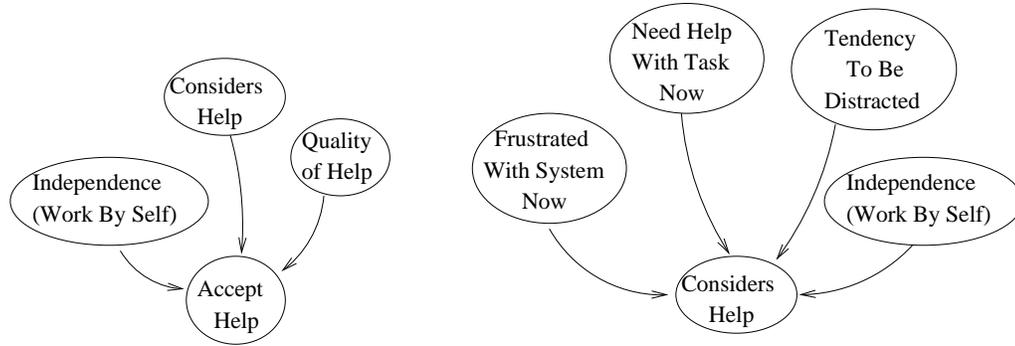


Figure 4.1: Influential factors. Left: Causes for accepting help. Right: Causes for considering help.

Figure 4.1 (right). Other factors can be modeled similarly.

Together, the variables $\{F, N, TD, TI\}$ make up the user's *state*. As we will see, these are sufficient to predict the probability of specific user behaviours (including accepting help) and how costly or rewarding a user perceives his experience with the automated system at any point in time. Recall the motivating examples we used at the beginning of the chapter: 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 partially correct suggestions that help 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. We discuss the precise structure of the reward and cost functions below.

Variables TD and TI are *static*, reflecting specific user traits that do not change over time. Naturally, these can change over certain time scales, but we take these to be static at least over the time frame associated with a reasonably small series of application sessions. In contrast, F and N are *transient*, reflecting user attitudes that can change, often frequently, during a specific 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* toward frustration and neediness in the application. These influence the (stochastic) evolution of F and N . We define a user’s *type* to be the state of all static user traits: $\{TF, TN, TD, TI\}$.

In our prototype application, these user variables are discrete, with variables F , N , TD and TI having three values each, and TN and TD having two values each. $F = 1$ 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. As a result, there are eighty-one user states and thirty-six user types.

4.3.1.2 Model Structure and Dynamics

Since the user state is partially observable, the system must maintain a probability distribution over user states given all past observations of user behaviour (reflecting the relative likelihood that the user is in a particular state). We refer to this belief state as $BEL(F, N, TD, TI)$, or BEL for short. Based on the current belief state, the system reasons about the rewards and costs of its actions in order to make an appropriate decision, and updates its beliefs after each user observation. A user’s type (over traits TF, TN, TD, TI), despite having a fixed value for a specific user, is not known *a priori* to the system, and thus, must also be estimated probabilistically.

The causal relationships in Figure 4.1 form the basis of our user model. In addition, the availability of automated assistance ($HELP$) affects when the user can consider suggestions. Our model incorporates additional system variables (SYS) and user observations (OBS). An example of a system variable is the status of a slider widget that allows the user to directly manipulate its settings. User observations should be abstracted at a behavioural level, and are useful for inferring the user’s state. Since these observations are domain-specific, we leave further discussion of the development of the observation model to Section 4.3.2.

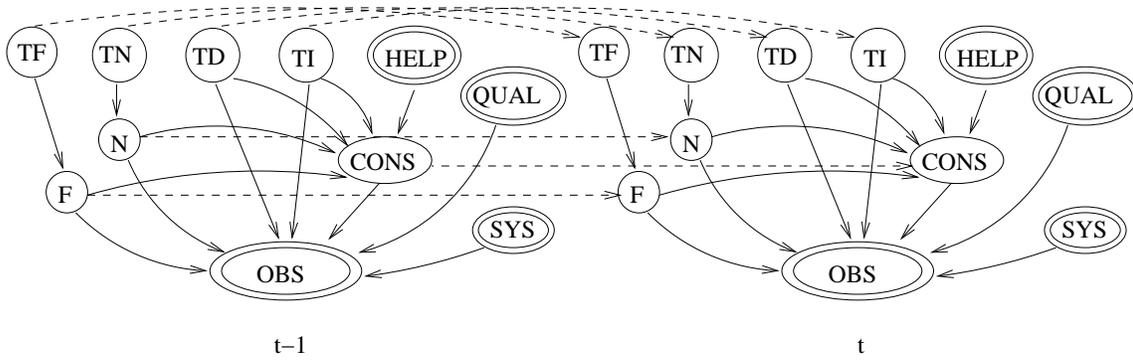


Figure 4.2: A two time-step DBN user model. Solid arcs indicate intra-temporal dependencies while dashed lines indicate inter-temporal dependencies. Observations are drawn as double ovals.

At a given point in time, the system observes the user’s action, infers the user’s current state, and decides whether to offer help at the next time step. Naturally, certain variable values may persist over time, or influence the values of other variables at the next point in time. For example, a user may be frustrated now, but over time, the frustration level will decrease if nothing else aggravates him. To model these temporal characteristics, we adopt a two-stage DBN model [DK89], as shown in Figure 4.2. In this model, variables F , N , and $CONS$ have temporal dependencies on their counterparts in the future, and the values of the user type variables persist over time. This model allows the system to learn the user’s type through behavioural observations.

Recall from Chapter 2 that a two-stage DBN models a joint distribution over a set of n random variables $\mathbf{X} = \{X_1, \dots, X_n\}$ at time $t-1$ and t . We denote the parameterization as θ , which specifies the set of conditional probability distributions, $Pr(X_i|Pa_i)$ for each X_i and its parents Pa_i . In particular, $\theta_{ijk} = Pr(X_i = x_i^k | Pa_i = pa_i^j)$ for the k^{th} value of X_i and the j^{th} parent configuration. We discuss these parameters in more detail in the next section.

The parameters of a DBN are defined by a prior distribution at time $t = 0$, a transition function, and an observation model. As an initial step, we handcrafted these parameters

using basic domain knowledge. The transition function for the user types is the identity function, since these variables represent persistent traits of a user. The transition functions $Pr(F_t|F_{t-1}, TF_t)$ and $Pr(N_t|N_{t-1}, TN_t)$ capture frustration and neediness patterns and how they evolve. The transition function for $CONS_t$ and the observation model for OBS_t are more complicated due to the size of the distributions. Here, we exploit their common substructure. For example, if help is not available, the user is not considering it, $Pr(CONS_t = not|HELP_t = none) = 1.0$. If help is available and the user was considering it ($CONS_{t-1} = yes$), then the probability of the user considering help now is defined as $Pr(CONS_t|F_t, N_t, HELP_t)$, which is independent of TD_t and TI_t . The intuition is that whether one will (dis)continue to consider help depends on changes in the levels of frustration or neediness. On the other hand, if the user was not considering help already, then the current consideration level will depend on the difficulty of the task and the user's tendency to work alone, $Pr(CONS_t|N_t, TI_t, HELP_t)$. The details of these parameters are provided in Appendix B.1.

As mentioned in Chapter 2, exact inference in DBNs can be done via the clique tree algorithm [HD96]. The performance of this algorithm depends on the size of the cliques which are created based on the dependencies in the DBN. In the context of user modeling, we are interested in monitoring the system's belief distribution over the user's state over time, given past observations: $Pr(BEL_t|OBS_{1:t})$. Let X_t denote the clique consisting of elements BEL_t , TF_t , TN_t , and $CONS_t$. Then the computation becomes:

$$Pr(BEL_t|OBS_{1:t}) = \sum_{TF_t, TN_t, CONS_t} Pr(X_t|OBS_{1:t}) \quad (4.1)$$

which we simplify as:

$$Pr(BEL_t|OBS_{1:t}) \propto \sum_{TF_t, TN_t, CONS_t} Pr(OBS_t|X_t)Pr(X_t|OBS_{1:t-1}) \quad (4.2)$$

Equation (4.2) corresponds to a *rollup* step in the clique tree inference algorithm. In Section 4.3.3, we discuss simulation results that gauge the speed and accuracy of this update process.

In order to estimate the user’s reaction to system actions, we must compute the likelihood of a user accepting help given its quality, the system environment, and past observations:

$$Pr(OBS_{t+1} = acc | HELP_{t+1}, QUAL_{t+1}, SYS_{t+1}, OBS_{1:t}) \quad (4.3)$$

We can estimate this term also using clique tree inference. Let Ev_{t+1} denote the set of observed variables $\{OBS_{t+1}, HELP_{t+1}, QUAL_{t+1}, SYS_{t+1}\}$. Then $Pr(Ev_{t+1} | Ev_{1:t})$ can be rewritten as follows:

$$Pr(Ev_{t+1} | Ev_{1:t}) = \sum_{X_{t+1}} Pr(Ev_{t+1} | X_{t+1}) Pr(X_{t+1} | Ev_{1:t}) \quad (4.4)$$

Since Ev_{t+1} and Ev_t are independent given X_{t+1} . Thus, we infer X_{t+1} at the next time step given the values of the potential help and system variables at $t + 1$ and marginalize OBS_{t+1} to predict the probability that the user will accept the potential system help. This term is used in the system’s decision making policy, which is described in Section 4.3.2.

4.3.1.3 Reward Function

In modeling a wide range of user types, we must consider multiple conflicting objectives: for general users, a level of independent functioning is considered desirable, so there is some cost to help; there is benefit of providing the right help when needed or desired; and there is a cost to providing incorrect suggestions, or suggestions when not needed or desired. Furthermore, the system should customize the degree of help based on its beliefs about the user’s current attitudes.

To evaluate automated help, we define a reward function and a cost function that incorporate user preferences toward automated assistance. The reward function depends on the user state and the quality of the suggestion, $R(F, N, TD, TI, QUAL)$ and is decomposed as follows: $R(F, TI, QUAL) + R(N, TI, QUAL) + R(TD, QUAL)$. This generalized additive decomposition reflects the assumption that the overall perceived value of help

(of some specified quality) can be determined by independent contributions given the current levels of frustration and neediness (each of these conditioned on degree of independence) and degree of distractibility. The cost of interrupting the user is defined as $C(F, N, TD, TI)$, irrespective of the quality of the automated help. We assume additive independence of the cost function: $C(F) + C(N) + C(TD) + C(TI)$. We normalize the range of the rewards and costs to be in $[-46,46]$. The precise parameters of this function are provided in Appendix B.1.

4.3.2 Test Application and Context of Use

In order to infer a user's state, we need to identify observations that correlate with those states. Therefore, the detailed structure of the model must be domain-specific. We chose a text editor as a test-bed application because it is familiar to many computer users and its functions are common to other communication software such as email and online chat. Furthermore, people with vocabulary and motor disadvantages often find that word processing and word prediction software allow them to concentrate on the quality of writing and give them a sense of authorship [HG00]. In the context of using a text editor, word prediction suggestions are viewed as automated help actions to assist the user in a series of typing tasks. The architecture of our system is presented in Figure 4.3. This architecture follows the DAISI framework as presented in Chapter 3, where the Interface, User Model, Language Model, and Decision Making Policy components correspond to the Application Interface, User Characteristics, User Goal, and Action Selection components respectively. Unlike other word prediction software, our system will not offer suggestions whenever a letter is typed. Rather, it learns the user's traits and needs, and makes suggestions only when it believes that the user can benefit from them. This methodology is generalizable to more complex software and tasks.

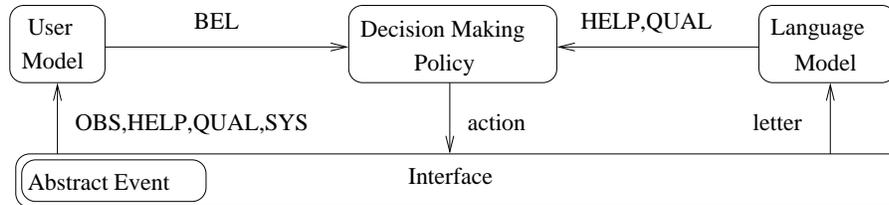


Figure 4.3: Overall system architecture.

4.3.2.1 Fully Observable Variables

In a typical computing environment, keyboard and mouse events are the only fully observable variables. We abstract these events into *behavioural* patterns that correlate with user characteristics. The resulting set of observations modeled in the variable *OBS* can be roughly categorized according to the various user state characteristics with which they are correlated:

- **Frustration:** continuously pressing a key down, moving the mouse back and forth quickly, jamming into the keyboard, multiple fast mouse clicks, explicitly indicating a need for fewer suggestions
- **Neediness:** erasing many characters, browsing (surfing menus, switching applications) for help, pausing
- **Distractibility:** browsing (surfing menus, switching applications) due to distraction, pausing
- **Independence:** explicitly indicating a need for more or fewer suggestions, accepting help/suggestions (as a function of quality)

Note that browsing and pausing are common to both neediness and distractibility, which is consistent with other proposed models [HBH⁺98]. This ambiguity creates additional uncertainty that the system needs to manage and further suggests the importance of a probabilistic model that account for multiple “causes” for observed behaviour. Other

user behaviours include responses to automated suggestions, such as accepting help (acc), hovering over the suggestion box (hh), and pausing when suggestions are present (hp). We also created a slider widget, *SDR*, which allows the user to explicitly indicate whether more or fewer suggestions are desired.

Under this design, the system has two actions — to offer a set of completion words (POP), or to remain passive (\neg POP). Altogether, there are 972 hidden states and 420 observations, yielding a total of 408,240 system states. The DBN model allows us to keep the representation compact in terms of the local distributions, rather than using a *flat* state representation (whose size is exponentially larger).

4.3.2.2 Language Model

The word prediction component is treated as a plug-in module in the system’s reasoning process. This module takes as input the previously typed word, w_{t-1} and the current prefix, w_{prefix} . As output, it returns a set of suggestions with a quality estimate (i.e., *QUAL*). This quality value is important because it directly impacts whether a user will accept automated assistance. Furthermore, in a word prediction domain, the quality of the predictions varies widely depending on the prediction algorithm used. These factors have a strong influence on the system’s decision whether to offer help as we will see below.

Standard word prediction software makes use of collocation statistics such as *n-gram probabilities* [SNN⁺01]. In particular, for $n = 2$, a *bigram* probability is defined as $Pr(w_t|w_{t-1})$. (In a prediction task, w_t must be consistent with w_{prefix} .) Our system also adopts a bigram model, which is trained on 40% of the 100 million word British National Corpus (BNC). The system maintains the top 20,000 bigrams and 20,000 unigrams with backoff weights in its lexicon at runtime. In addition to using the bigram probabilities, we want the suggestion feature to offer completion words that are *different* from each other. In other words, we want the completions to cover a larger probability mass. For example, with $w_{t-1} = \text{“the”}$ and $w_{prefix} = \text{“nu”}$, a bigram model may offer suggestions

“number”, “numbers”, and “nuclear” even though “number” and “numbers” only differ by one letter. Therefore, we propose a similarity metric that captures the *expected savings* a word provides to the user.

At a given point in time, there is a set of $\{c_1, \dots, c_K\}$ words that are plausible (i.e., non-zero probability) completions given w_{t-1} and w_{prefix} . Each c_k has an associated bigram probability, p_k . To attribute a utility measure to a suggestion, we first define its utility with respect to a true word s defining $U(c_k|s)$ to be the number of identical prefix characters less the number of characters erased less the number of characters added to change a completion word, c_k , into the true word, s [FLL02a]. For example, $U(\text{“are”}|\text{“all”}) = 1 - 2 - 2 = -3$, while $U(\text{“apples”}|\text{“apple”}) = 5 - 1 - 0 = +4$. Given $U(c_k|s)$, we define the *expected savings* of a completion suggestion c_k as:

$$ES(c_k) = \sum_{i=1}^K U(c_k|c_i)p_i \quad (4.5)$$

where p_i is c_i 's bigram probability. We define the *joint* expected savings of a set of completion words as:

$$JES(c_1, \dots, c_J) = \sum_{i=1}^K \operatorname{argmax}_{c_j} U(c_j|c_i)p_i \quad (4.6)$$

for a suggestion with J words. The intuition is that, for any true c_i , the user will choose from among the J suggested completion words and accept the suggestion offering maximum savings. In the example with $w_{t-1} = \text{“the”}$ and $w_{prefix} = \text{“nu”}$, the JES model chooses the suggestions: “number”, “nuclear”, and “nurses”.

Unfortunately, when $J \geq 2$, the number of comparisons increases exponentially. We propose a greedy implementation for our *JES* model. First, among K words, choose $s_1 = \operatorname{argmax}_{c_k} ES(c_k)$. With $K - 1$ words left, choose the second best completion with respect to s_1 ; that is, $s_2 = \operatorname{argmax}_{c_k} JES(s_1, c_k)$; and so on. This greedy approach results in $O(J - 1 \cdot K)$ comparisons. The estimated quality of a suggestion is simply its joint expected savings.

Table 4.1: Comparison of prediction techniques

Method	Avg (sec)	Max (sec)	Exact	Substr	Util
BIGRAMS	0.1485	0.9693	3629	5674	9151
GREEDY	0.1743	1.0285	3578	5706	9740
ENUM	0.7110	3.5008	3708	5806	10169

Table 4.1 shows a comparison of these algorithms through experiments implemented in Matlab and run in Linux with 3.60G Hz CPU, with $K = 40$ and $J = 3$. We use bigrams (BIGRAMS) as the baseline comparison by taking the J most probable words, and we show the speed performance of using JES for word prediction, implemented both greedily (GREEDY) and by full enumeration (ENUM). We ran these three algorithms through a text of length 11,718 characters (with 7917 word prediction opportunities). We see that the average and maximum times for the BIGRAMS and GREEDY techniques are similar, while ENUM is too slow for an online task. We kept track of the number of correct predictions made (Exact), the number of predictions that contained a substring of the true word (Substr), and the actual character savings (Util). The JES greedy implementation scores almost as well as the bigram model on correct predictions, but this is not our main concern. Critically, the JES model provides much greater utility with respect to character savings. It is also significantly faster than full enumeration but still offers acceptable performance with respect to utility (note that ENUM provides optimal suggestions). Results from the usability experiments in Section 4.3.5 also suggest that the expected savings metric is more helpful for users.

4.3.2.3 Decision Policy

The decision problem faced by the help system is characterized by considerable uncertainty. Obviously, the word a specific user is typing cannot be predicted with certainty, though the language model allows us to quantify this probabilistically and rather pre-

cisely. More importantly, whether a user could benefit from the system’s help, or desires such help, cannot be assessed with certainty either. Our model is designed to (probabilistically) predict whether a user needs or wants help based on past observed user behaviour.

We define a myopic policy that models the uncertainty and systematically trades off the conflicting objectives as follows. At each time step, the system takes an action and the user can either accept it ($OBS = acc$) or not ($OBS = \neg acc$). Considering these possible outcomes, the *expected utility* of an action is:

$$EU(POP) = EU(POP|acc)Pr(acc) + EU(POP|\neg acc)Pr(\neg acc) \quad (4.7)$$

If the user accepts a suggestion, the system will “receive a reward” reflecting the net benefit of the suggestion (incorporating any costs of interruption, etc.). Of course, the system can only compute the *expected* reward since the user state is not fully known. Thus, we define the expected utility of a suggestion given that the user accepts help as:

$$EU(POP|acc) = \sum_{F,N,TD,TI} R(F, N, TD, TI, QUAL)BEL(F, N, TD, TI) \quad (4.8)$$

On the other hand, if the user rejects the suggestion, the system will receive a penalty, again, in expectation given the user’s type. The corresponding expected utility is:

$$EU(POP|\neg acc) = \sum_{F,N,TD,TI} C(F, N, TD, TI)BEL(F, N, TD, TI) \quad (4.9)$$

We predict the value of making a suggestion by taking the expected value of POP with respect to the probability of acceptance. The overall system policy is to take the action with the maximum expected utility: *pop up a suggestion if $EU(POP) > EU(\neg POP)$* , where not taking any system action has zero utility, i.e., $EU(\neg POP) = 0$.

4.3.3 Simulation Results

To assess our user model, we ran text editing simulations with word prediction. The test text consisted of sentences drawn randomly from 10% of an unseen portion of the BNC.

We sampled from a simulated user model based on the DBN described in Figure 4.2.

For each user type, we ran 100 trials with texts about 330 words long. The averaged results show that the system’s beliefs converged to the true type in all thirty-six cases. (Convergence was determined using a threshold of $p \geq 0.85$, where p is the system’s belief of the true user type.) The time it took the system to reach convergence varied from 130 to 620 observations. At an average rate of approximately 19.35 observations per word typed, convergence was reached between 6.7 and 32.0 words typed. On average across all user types, the system’s belief converged after 216 observations (or, approximately 11.2 words typed). Examples of convergence curves for three different user types (as a function of the number of observations over time) are shown in Figure 4.4. For example, Figure 4.4 (left) shows the results for inferring Type 2, whose specific values are $\{TD=1, TI=1, TN=1, TF=2\}$ (shown on the y-axis of the subgraph). In the beginning of the interaction, we see from this plot that between the time when 0 to 100 observations have been made, the system also considers an alternative user type $\{TD=1, TI=1, TN=2, TF=2\}$ to be likely. Since this alternative user type only differs in the tendency to be needy, the legend in the subgraph shows that the alternative user has that variable’s value changed to $TN=2$. Soon after the start of the interaction, the system is able to distinguish this user type from Type 2. Similarly, Figure 4.4 (middle) shows the results for inferring Type 6, whose specific values are $\{TD=2, TI=1, TN=1, TF=2\}$, in comparison to two competing user types differing in the value of TD and TI (their specific values are indicated in the legend). Here, we see that the system takes longer to distinguish between the true user type and one who differs in the value of TD . Lastly, Figure 4.4 (right) shows the results for inferring Type 33, whose specific values are $\{TD=3, TI=3, TN=1, TF=1\}$, in comparison to two competing user types differing in the value of TD and TN . In this case, the system displays more difficulty in isolating the true user type.

In our model, we chose an abstract representation of behavioural observations that is

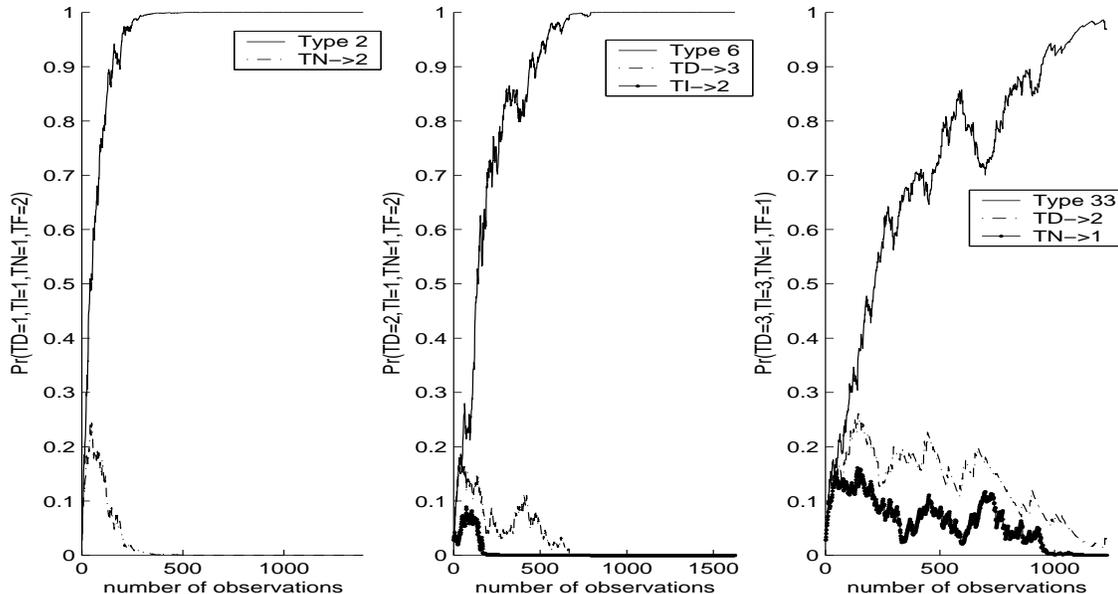


Figure 4.4: Examples of belief monitoring. Left: early convergence. Middle: convergence with respect to competing types. Right: slow convergence.

intuitive from a designer’s perspective and that reduces the number of temporal dependencies (see the discussion in Section 4.3.2). Belief state monitoring was implemented in Matlab 6.5 R13. On average, this computation takes approximately 0.57 second on a Pentium M, 1.2G Hz CPU, 386 MB RAM processor. This prototype implementation can be considerably accelerated, so real-time inference is not a concern in this task. When observational abstraction of the type used here is not feasible, or does not provide enough decomposition of the inference task to allow real-time belief state monitoring, approximation algorithms for monitoring (e.g., [BK98]) can be considered. Furthermore, belief update based on aggregate observations (e.g., every k steps) can also be used; since user state will generally evolve on much longer time scales than individual observational events, slight lags in user state estimation will generally have a negligible effect on performance.

A system’s overall utility is quantified in terms of the rewards and costs it receives during its interaction with the user. In Section 4.3.2, we defined implicit reward and cost

functions that vary according to the user’s state in Section 4.3.1. Here, we use them to define the overall utility given the user state and the actual quality of the suggestions, $U(F, N, TD, TI, QUAL)$:

$$U = \begin{cases} 0 & \neg\text{POP} \\ R(F, N, TD, TI, QUAL) & \text{POP and OBS} = \text{acc} \\ 0 & \text{POP and OBS} = \text{hh, hp} \\ C(F, N, TD, TI) & \text{POP and OBS o/w} \end{cases} \quad (4.10)$$

Generally, Equation (4.10) states that the system receives a reward when a suggestion is accepted and a cost when a suggestion is rejected. The user’s response to the system suggestion is used as feedback to help infer the type of user that the system is dealing with. Thus, based on observed user behaviour and reactions to system help, the system infers the user’s type and adapts its interactivity level for that user. By plotting the rewards and costs over an interaction, we see different kinds of “reward patterns” that reflect the system’s adaptivity to the inferred user type. In Figure 4.5, we provide some examples of reward patterns received by the system in three cases. The leftmost graph shows that the system made suggestions continuously, with most of them resulting as rewards (i.e., user accepting help) and some of them resulting as costs (i.e., user rejecting help). While this pattern is taken from the user type with $\{TD=1, TI=2, TN=2, TF=1\}$, other user types with lower levels of independence and/or a combination of high level of tendency to be needy but low tendency to be frustrated and low distractibility also exhibit similar adaptive patterns. The middle graph shows a similar reward pattern, with clearly fewer suggestions made by the system. This pattern was taken from the user type with $\{TD=1, TI=3, TN=2, TF=2\}$. The rightmost graph shows that the system made some suggestions early on, but stopped offering help after about 400 observations. This pattern was taken from the user type with $\{TD=2, TI=3, TN=1, TF=2\}$. Other types with high levels of independence and distractibility also exhibit similar patterns.

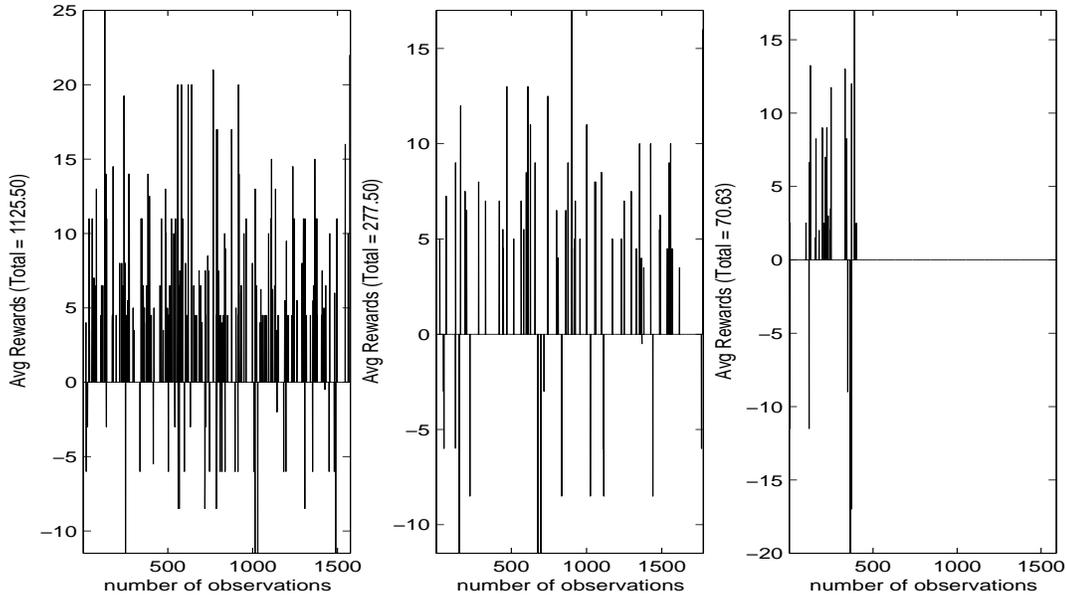


Figure 4.5: Examples of system behaviour according to inferred user type.

These examples illustrate the differences in the system’s belief about the user and the associated system behaviour. As we discuss in Table 4.2 below, 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. The policies we chose for this comparison are: suggest only if the quality is greater than a threshold (THRESH , for $QUAL > 3$), always make suggestions (ALWAYS), and never make suggestions (NEVER). We refer to our system policy as MEU because it considers the maximum expected utility of actions. Table 4.2 compares the average reward per time step for these different policies with respect to some illustrative user types. The full set of results for all 36 types is included in Appendix C.1. Generally, ALWAYS results in some negative cost on average, because it is unable to account for the user type or the evolving user state at hand. In particular, with dependent users who need help, always offering suggestions result in a low cost, as we see in the first case with Type 3 in the table. However, it does poorly (often extremely) in other cases. Overall, the

various user types show that the system performs better using adaptive policies. In the cases where the system need to make tradeoffs regarding the various user variables, the system benefits more by using MEU as the adaptive policy across the different user types. This is because the MEU policy will provide suggestions to users who are likely to want more suggestions, and back off from offering suggestions to users who do not desire help. These cases illustrate that a static policy, such as ALWAYS and NEVER, or a policy that disregards the user type, such as THRESH, suffers most. Overall, MEU performs better than THRESH for seventeen of the thirty-six user types, with better performance mostly for highly independent users; for the remaining cases, MEU and THRESH perform comparably (within 0.25 to 2.5 of each other). Although NEVER receives zero rewards in all the cases, it is unable to detect cases when the user in fact needs help, which is a state that could change from time to time.

To see the impact of the system learning about the user over time, we show the results using only the last 10% and the last 2% of the simulation data from each trial in Table 4.3 and Table 4.4 respectively. For space reasons, we show some sample user types here, but the full set of results for all the user types are reported in Appendix C.2. From these tables, we see in general that the performance for ALWAYS, THRESH, and NEVER are fairly similar to the overall performance reported in Table 4.2 above, while the performance for MEU improves. Moreover, the average rewards for MEU are higher later on in the trials. Specifically, in the last 10% of the data, MEU has five user types that have positive average rewards, nine user types that have zero average rewards (the system learned to not offer help in these cases), and it beats THRESH in twenty-seven of the thirty-six user types. Similar improvements are evident in the last 2% of the data: MEU has fifteen user types that have positive average rewards, ten user types that have zero average rewards, and it beats THRESH in twenty-six of the thirty-six user types. These results indicate the system is able to learn suitable suggestion behaviour for different user types.

Table 4.2: Comparison of policies using average rewards by user profile {TD, TI, TN, TF}

User Type			ALWAYS	MEU	THRESH	NEVER
3	{1,1,2,1}	min, max	-19, 46	-19, 46	-19, 46	0, 0
		avg	-1.78	0.90	2.45	0
		stderr	0.6016	1.2492	1.2135	0
13	{1,2,1,1}	min, max	-22, 45	-22, 45	-22, 45	0, 0
		avg	-4.71	-1.05	-0.72	0
		stderr	0.5518	1.8969	1.3593	0
17	{2,2,1,1}	min, max	-25, 40	-25, 40	-25, 40	0, 0
		avg	-8.92	-5.68	-5.07	0
		stderr	0.3337	0.8886	0.9980	0
21	{3,2,1,1}	min, max	-32, 38	-32, 38	-32, 38	0, 0
		avg	-15.30	-7.48	-11.86	0
		stderr	0.4425	4.7051	1.0875	0
25	{1,3,1,1}	min, max	-33, 44	-33, 44	-33, 44	0, 0
		avg	-11.03	-5.10	-4.50	0
		stderr	0.8988	1.2277	1.6504	0
26	{1,3,1,2}	min, max	-33, 44	-33, 44	-33, 44	0, 0
		avg	-16.06	-8.30	-10.84	0
		stderr	0.7687	2.2112	1.0073	0
34	{3,3,1,2}	min, max	-43, 37	-43, 35	-43, 37	0, 0
		avg	-26.66	-0.41	-21.69	0
		stderr	0.6841	23.9873	1.8065	0
Average			-12.93	-5.35	-8.73	0

4.3.4 Learning Model Parameters

To replace the handcrafted parameters in the user model, we designed controlled experiments that explore different user states and logged corresponding user behaviour.

Table 4.3: Comparison of policies using the last 10% of data for average rewards by user profile $\{TD, TI, TN, TF\}$

User Type			ALWAYS	MEU	THRESH	NEVER
3	$\{1,1,2,1\}$	avg	-1.64	0.54	2.44	0
13	$\{1,2,1,1\}$	avg	-4.52	-0.52	-0.69	0
17	$\{2,2,1,1\}$	avg	-9.10	-3.74	-5.24	0
21	$\{3,2,1,1\}$	avg	-15.70	4.23	-12.39	0
25	$\{1,3,1,1\}$	avg	-10.88	-3.48	-2.62	0
26	$\{1,3,1,2\}$	avg	-15.97	-7.35	-10.54	0
34	$\{3,3,1,2\}$	avg	-26.52	0.00	-22.65	0

Table 4.4: Comparison of policies using the last 2% of data for average rewards by user profile $\{TD, TI, TN, TF\}$

User Type			ALWAYS	MEU	THRESH	NEVER
3	$\{1,1,2,1\}$	avg	-1.98	0.87	3.47	0
13	$\{1,2,1,1\}$	avg	-5.04	2.14	-1.16	0
17	$\{2,2,1,1\}$	avg	-8.60	0.95	-4.68	0
21	$\{3,2,1,1\}$	avg	-15.60	-1.10	-12.66	0
25	$\{1,3,1,1\}$	avg	-12.17	0.76	-1.74	0
26	$\{1,3,1,2\}$	avg	-17.35	-0.15	-6.48	0
34	$\{3,3,1,2\}$	avg	-27.14	0.00	-16.63	0

Because user states are not directly accessible and cannot be explicitly elicited at every time step, our experiments collected data in a semi-supervised fashion.

4.3.4.1 Data Collection Experiments

Since most potential participants can type quickly without help, we designed a procedure that requires the user to type with a Dvorak keyboard in a simple text editor as shown

in Figure 4.6. Note that the right hand side of the text editor has a slider control that allows the user to explicitly indicate whether he wants more or fewer suggestions. This slider widget is akin to the “volume control” suggested in Lumière [HBH⁺98].

There were forty-five users and each participated in three conditions. In all the conditions, users were given a print out of text and asked to type it in to the text editor as accurately as possible. The first condition introduced artificial delays and “sticky keys” into the system at fixed intervals. In particular, delays were set between 2-5 seconds, and sticky keys have an effect of inserting between 2-5 extra characters with a key press. The second condition presented a mix of audio and visual pop-up distractors at regular intervals. These distractors have a lifetime of seven seconds and can end earlier if the user closes their residing windows. In the third condition, the text to be typed by the user contains a higher percentage of long words and esoteric vocabulary, as measured using the Fog index [Gun68]. The first two conditions used text with Fog index = 11, while the third used Fog index = 30. To assess the user’s current state, questions to elicit the user’s current F and N values were posed at the end of each sentence in all the trials. A post-questionnaire was designed to assess the user’s general attitudes and tendencies under this computing environment so that we could elicit the user’s type. This questionnaire is provided in Appendix A.1.

Based on informal observations, we identified a wide range of behaviours. For example, some participants ignored pop-up animations and audios, some laughed at them, while a few explicitly closed them. Participants also varied in their strategies for dealing with word completion suggestions. Some typed one letter at a time while anticipating a suggestion and accepted it when it appeared, some buffered a few characters, typed them, and watched for suggestions, while others just did not accept the suggestions at all. Frustration was either not shown or appeared as a pause or sigh, but rarely as a physical action. We suspect this subtlety is influenced by the controlled environment and the presence of a researcher.

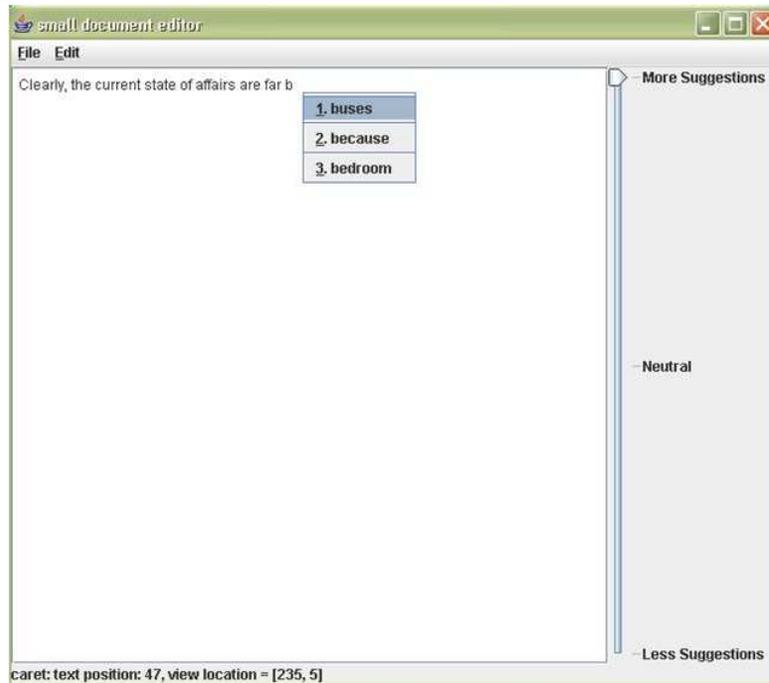


Figure 4.6: The Java prototype with word completion suggestions.

4.3.4.2 Parameter Estimation

The learning task at hand is one with known Bayesian network structure with incomplete data. With forty-five participants and three sequences of observations each, there are a total of $M = 135$ training cases. Each sequence on average consists of 845 observations, but ranges from 99 to 3097. Our goal was to learn the prior distributions F_0 , N_0 , $CONS_0$, the transition functions F_t , N_t , $CONS_t$, and the observation function, OBS_t . We applied a standard algorithm, expectation-maximization (EM) [DLR77, Mur02]. The initial parameters were set randomly. EM iterates between computing the expected values of the hidden variables (given parameter estimates) in an E-step and maximizing the parameters given the data in an M-step, as follows:

- **E-step:** given $\hat{\theta}$ and data set $D = \{\mathbf{y}_l\}$, compute:

$$E_{Pr(\mathbf{x}|D, \hat{\theta})}(C_{ijk}) = \sum_{l=1}^M Pr(x_i^k, pa_i^j | \mathbf{y}_l, \hat{\theta})$$

- **M-step:** given the sufficient statistics, compute:

$$\theta_{ijk} = \frac{\alpha_{ijk} + E_{Pr(\mathbf{x}|D,\hat{\theta})}(C_{ijk})}{\sum_{k=1}^{r_i} (\alpha_{ijk} + E_{Pr(\mathbf{x}|D,\hat{\theta})}(C_{ijk}))}$$

where C_{ijk} is the number of times x_i^k and pa_i^j occur in the data set and α_{ijk} is a bias on the corresponding θ_{ijk} .

EM is guaranteed to converge to a local minimum. To avoid the sparse data problem and to incorporate prior knowledge, we used the handcrafted parameters in the simulations as biases in calculating a maximum a posteriori estimate in the M-step. In particular, we tried different weightings of the data and biases. The weights we used are: 0% of priors (i.e., data only), 10%, 30%, and 50%. We report on the training results using different weights in the next section.

4.3.4.3 Model Comparison

In this section, we describe the procedures for discretizing the user variables and learning the parameters of the DBN.

The post-questionnaire in our experiments elicited the values of TF , TD , and TI using nineteen items, with each item intending to elicit a particular variable. Sample questions asked the user to self-report on a Likert scale whether they felt frustrated with the sticky keys, whether they were distracted by the pop-up animations, or whether imperfect suggestions were selected. Since this questionnaire was newly designed for this experiment, we carried out factor analysis on the responses to identify possible clusters and underlying factors [Gor83]. Due to the small sample size of forty-five, this analysis is a preliminary step in checking for strong correlations only. According to the Kaiser criterion, four factors had eigenvalue higher than 1.0; the scree test indicates three to seven factors; using the percentage of variance explained, we obtain three factors for 74%, four factors for 84%, and five factors for 93%. Finally, we retain three factors by the interpretability criterion so that one factor corresponds to one design variable.

We used variance maximizing rotation to extract principal components. The resulting factor loadings confirmed that ten items clustered with the intended factor, two items clustered incorrectly, and seven items were undetermined. The incorrect items were reclassified into their clustered factors while the others maintained their original classes. Thereafter, we used the responses to compute the participants' score for TF , TD , and TI as $\sum_{i=1}^l \frac{r_i}{l}$, where r_i is a score and l is the number of items in the factor. By inspection, we partitioned the results into the domain of our model variables, i.e., two categories for TF and three for each of TD and TI .

For TN , we used typing speed (spd) as the motor attribute and the percentage of unfamiliar vocabulary ($vocab$) as the cognitive attribute in a user's neediness level. Let f_1 and f_2 be the normalized factor loadings for spd and $vocab$ respectively. Then $TN = f_1 \times spd + f_2 \times vocab$. By inspection, we partitioned the results into two categories for TN .

Based on this procedure, each participant has a user type profile $\{TF, TN, TD, TI\}$ capturing their general tendencies in a computer setting. We plotted these profiles in Figure 4.7 to identify the types of users in our participant pool. As shown, our pool did not cover all thirty-six possible user types in our model — the reason could be due to the small sample size or that some types simply do not exist. In many cases, we had one or two participants of a type (e.g., $\{1,2,2,1\}$). For type $\{2,2,3,3\}$, five participants had this profile. This plot suggests that users who are highly independent ($TI=3$) tend to get frustrated ($TF=2$) by the system. It also suggests a correlation between dependent and needy users (with TN negatively correlated to TI).

As mentioned earlier, we used different weightings of the data and handcrafted parameters to train our model. If given enough representative data, we could compare these results using cross validation. However, our data set is very small relative to the state space so we discuss our choices informally.

With respect to the distributions using handcrafted parameters, we computed the

	TD=1		TD=2		TD=3	
TI=1	TF=1	3	1		1	2
	TF=2	1	1	2		1
TI=2	TF=1		1	2	1	2
	TF=2	1	1	2	4	
TI=3	TF=1	1	1			
	TF=2		4	2		3
	TN=1	TN=2	TN=1	TN=2	TN=1	TN=2

Figure 4.7: The topology of our participants. Each box indicates the number of participants belonging to that user type.

Kullack-Leibler (KL) divergence [KL51] to assess the relative entropy of the learned distributions using $KL(P||Q_w) = \sum_{x \in X} P(x) \log \frac{P(x)}{Q_w(x)}$, where P is the handcrafted distribution and Q_w is the learned distribution trained with weight w (the weight of the handcrafted parameters). The maximum KL divergence shows little difference among them: 3.1455 for the prior for *CONS*, 0.015 for the prior for *N*, and less than 0.01 for the others. If we had greater differences we could compare their performance further. However, with our results, we chose to use the learned distributions trained with $w = 10\%$.

4.3.5 Usability Experiments

We designed a usability experiment to better assess the applicability of the learned model with real users by comparing their preferences with other system policies. We adopted a similar computing environment to the one used in data collection experiment (cf. Section 4.3.4). In addition, the inference engine necessary to update the user model was implemented in Matlab. Thus, a Matlab-Java server was implemented so that the Java

interface connects to it as a client.

In this experiment, the user’s task was to copy ten sentences from paper into our editor using a Dvorak keyboard. These sentences were taken from an unseen test set (Fog index = 15). There are four conditions in this experiment, each employing a different system policy. The four policies we chose for this initial comparison are those used in the simulations (cf. Section 4.3.3): THRESH, MEU (our system), ALWAYS, and NEVER. Participants were asked to type as accurately as possible. A questionnaire was given at the end of each condition and at the end of the entire experiment. In total, we had four participants. Due to the small number of participants in this study, no statistical hypotheses were tested.

Since the users in this pilot study are all novices with Dvorak (typing speeds between four to eight words per minute), they all preferred having as much help as possible. Contrary to other findings [FLL02a], one user commented that the system should provide completions even for short words like “and” and “to”. We suspect that if we had users with a wider variation in typing speeds (i.e., different levels of neediness, TN), the results would reveal greater differences in their preferences.

Three users preferred ALWAYS to both adaptive strategies, which were in turn preferred to NEVER (i.e., “the more help the better”). This pattern is supported by the average percentage of characters typed using the four policies, as well as the subjective responses to whether a particular policy helped reduced effort and time. The actual time, however, revealed that typing with NEVER was the fastest, followed by the two adaptive strategies, and ALWAYS was the slowest. The fourth user preferred the two static strategies equally over the adaptive ones, because he could not predict exactly when the suggestions would appear. Preference for predictable system behaviour is a common theme in interface design. We can model these preferences as user features in the state space to reflect different attitudes toward usability objectives. In this way, the cost model can weigh the amount of disturbance each action imposes on the user’s

mental model of the application. In the next section, we turn to the development of such a model in the context of a menu selection task.

Unfortunately, the Matlab-Java engine had computational overhead (that created extra threading and file I/O delays). While there was no inherent problem in the model itself that caused these delays, we found that the overhead caused MEU to be ranked lower than the alternative adaptive policy for two users. They added that if the system were faster, they would have preferred it over THRESH. All four users noted the quality of the suggestions in MEU was notably better than those in THRESH, although in three of the four cases, the percentage of correct suggestions were higher in THRESH. This suggests that the users perceive the utility of character savings as part of the quality of a suggestion. Also, the percentage of acceptances were higher in THRESH. This behaviour indicates they are dependent users in our model. Indeed, from plotting the system's inferred belief states, all four users were inferred as dependent ($TI = 1$) and needy ($TN = 2$). The number of characters typed using MEU was on average lower than those using THRESH.

Finally, about 20% of all the acceptances were partially correct suggestions, where users accept non-exact words and erase the endings. This indicates that the utility metric used in our language model (cf. Section 4.3.2) is more helpful than one that only uses a bigram statistic.

Despite the overall ranking by the users, which suggests that the adaptive strategies were not as useful as the static strategies, overall we find the results quite encouraging. Apart from a time-delay artifact, and concerns about predictability, user comments suggest that utility-based help is more desirable than suggestions made purely based on probability of acceptance. The task of typing using an unfamiliar keyboard layout was also perhaps more difficult than anticipated, leading to a bias for ALWAYS, which may not be present over time (as skill levels improve) or in less unfamiliar tasks.

4.4 Characteristics that Influence Disruption

This section presents the second case study where we focus on developing a probabilistic representation of the user’s mental model in the User Characteristics Component of the DAISI framework [HB06a, HPB09]. Through repeated interaction, a user builds a *mental model* of the application that reflects the knowledge gained through experience. This may include available software functionality, the locations of functions, the effects of those functions (e.g., how long they take, whether multiple actions achieve the same result), etc. A key bottleneck in building effective adaptive systems is accounting for the induced *disruption* to a user’s mental model. Consider, for instance, menu selection. The first time a user selects `Exit` from the `File` menu, he scans all the items inside `File`. After using `Exit` several times, he learns it is located at the bottom of `File`. An adaptive system may observe `Exit` is frequently used, and decide to move it to the top of `File` so that future access becomes faster. Obviously, there are tradeoffs involved: there are long-term task performance gains (the user will access the frequently used `Exit` more quickly); however, the disruption of the user’s mental model of `Exit`’s location may cause short-term performance degradation — more search is involved until the new location is learned — and annoyance. The degree of disruption is, intuitively, related to the strength of the user’s prior beliefs and the degree of “new search” required (e.g., if the new location is near the old one, disruption may be less than if it were further away).

While notification mechanisms may ease the abrupt transition caused by adaptive actions, some users may find them distracting or unnecessary. Ideally, an adaptive system should assess a user’s mental model, and make the tradeoffs between the *long-term* benefits of adaptive actions (e.g., improved task performance) and the costs of disruption before taking those actions. We propose just such a model here, with a focus on models of function location, relevant to adaptive systems that change function locations in order to make access more convenient or to reduce interface bloat. Our mental model is probabilistic: it allows for a natural definition of strength, model dynamics (including

learning and forgetting), and cost of disruption. We also propose means for assessing the long-term tradeoffs in a decision-theoretically principled fashion. This stands in contrast to adaptive system models that focus only on maximizing benefits of speed performance, while designating a “generic” cost to adaptive actions or ignoring costs altogether [HGIB08]. Although we focus on adaptive menus, our mental model representation and decision-theoretic approach applies more broadly.

We first review well-established properties of the mental model in Section 4.4.1. In Section 4.4.2, we propose a probabilistic approach that explicitly models the user’s mental model and the cost of disruption induced by the system’s adaptive actions. This approach allows an adaptive system to tradeoff disruption cost with expected savings (or other benefits) caused by a potential system adaptation in a principled, decision-theoretic fashion. We present this system in Section 4.4.3 in the context of menu selection tasks and demonstrate its performance in Section 4.4.4 in comparison to other static and adaptive approaches. We conducted two experiments with forty-eight participants to learn model parameters, which we discuss in Section 4.4.5 and Section 4.4.6. Lastly, Section 4.4.7 presents a usability experiment with eight participants. These results suggest that our approach is competitive with other adaptive menus with respect to task performance, while providing the ability to adapt to user preferences, and better support for reducing disruption.

4.4.1 Properties of Mental Models

The term *mental model* has been used in psychology to denote a person’s mental representation of (concrete or abstract) objects [JL83, GS83]. We use the term in its HCI sense, pertaining to software usability—a mental model is a user’s representation of an application. In particular, we focus on mental models of function locations, e.g., the location of menu items. Much research on mental models describes conditions and effects, but does not offer explanations or theoretical predictions [RW92]. Furthermore, due to

variations in the elicitation and experimental procedures, much debate surrounds these results [CO87].

Despite the imprecise state of work in the area, most researchers agree that people's mental models are *incomplete*, *dynamic*, and *unstable* [Nor83]. Consider a word processing application with standard menus like **File**, **Edit**, etc. A typical user will learn where common functions are located within menus, but will not know the locations of all available functionality, thus rendering his mental model incomplete. Through extended usage, the user reinforces, or strengthens, what he already knows and learns new information about the application. Certain knowledge may weaken as well (e.g., rarely used functions). This illustrates the dynamic nature of a mental model. Finally, if we ask the user to report his beliefs (based on his current mental model), elicited responses will often vary, even without disruption to the mental model, indicating the instability of mental models.

In HCI, mental models are often used to account for software *learnability*, or the user's ability to master the application. For example, after usage, a researcher may elicit/evaluate how well a user's mental model depicts the actual state of the software. If a system could "track" a user's mental model at runtime, it could identify sources of disturbance to the mental model and eliminate them from the design of the software. To this end, some computational approaches to mental models have been proposed, but are non-probabilistic and use heuristic updates [Sas97]. Thus, these approaches do not account for the three properties above in a principled manner.

Our aim is to develop an approach to mental models that allows a system to quantify the disruption caused by adaptive actions. We are unaware of any such formal models of disruption. In the next section, we present a probabilistic representation that adequately captures the three mental model properties. To make our discussion more concrete, we consider three types of adaptive actions in menu selection tasks (although the general principles apply more broadly): moving a function to the top of the menu and shifting

functions down as needed (TOP); swapping a function with the one above it (SWAP); hiding a function at the bottom of a menu (e.g., under double arrows) and shifting functions as needed (HIDE); and the non-adaptive default of not moving any items (NONE). We aim to capture the effect of such actions on a user’s mental model in order to quantify the induced disruption.

4.4.2 A Mental Model of Function Location

Most mental model research elicits information to determine the representation of mental models, “rather than trying to determine which set of hypothetical mental operations is supported by the data” [RW92]. Here, we develop a probabilistic representation of a mental model for function location that supports three key operations of interest for adaptive systems: learning, forgetting, and modeling disruption.

4.4.2.1 Basic Model and Model Strength

Let K be a set of possible functions each located in one of L (menu or interface) locations. For any $k \in K$, the user’s mental model of k ’s location, θ^k , is a multinomial distribution over L locations. We might think of $\theta^k(l)$ as the probability the user will attempt to (first) access function k at location l . As adaptive systems may “copy” the same function to allow access from multiple places of the interface, this probabilistic representation naturally models the possibility that k is accessible from multiple locations via a multimodal probability distribution. While generally not true, we assume for simplicity that the distributions $\theta^1, \dots, \theta^K$ are independent, allowing a convenient marginal (rather than joint) representation.

Without experience, model θ^k will be “weak” (e.g., uniform over L or a prior induced from using similar interfaces). Using k then generally requires the style of search typical of novices (e.g., visual search) [HK97a]. After executing k some number of times, we expect the model θ^k to become stronger (and more accurate), with higher probability assigned

to the true location (and possibly nearby locations, spatially or analogically), leading to reduced search time, and eventually leading to expert behaviour (e.g., logarithmic search or “immediate” execution) [CGG07]. Indeed, *strength* of a mental model, or how well users believe they know a function’s location, seems to be a key criterion in how they assess their own level of interface expertise. Our model leads to a natural notion of strength, which we use extensively below to quantify disruption. In what follows, we focus our formalization on model strength, rather than the model itself.

Model strength is defined with respect to a user’s degree of uncertainty or normalized entropy of the model distribution. Specifically, the strength of θ^k is defined as:

$$M^k = 1 - \frac{H(\theta^k)}{H_L^+} \quad (4.11)$$

where $H(X) = -\sum_x P(x) \log(P(x))$ is the entropy of distribution X , and H_L^+ is the maximum entropy of any multinomial with L events (i.e., the entropy of the uniform distribution over L locations). This definition normalizes strength in the interval $[0, 1]$, with 0 corresponding to the “weakest” mental model (high normalized entropy, no knowledge of function location) and 1 corresponding to the “strongest” mental model (low normalized entropy, complete certainty of location).

Using this representation, we develop the dynamics of mental models, with an eye toward quantifying disruption cost, using three key operations: learning, forgetting, and disruption due to adaptation. (The first two apply to any form of software, adaptive or not.)

4.4.2.2 Learning

We begin by examining how location models are learned. As noted, we expect model θ^k to become stronger with increased usage of k . Additional factors may also influence this process. For example, using a function k' located *near* k may reinforce θ^k somewhat—searching for k' may involve seeing (and learning) k ’s location. Thus, usage history of k



Figure 4.8: Learning and changes in the mental model distribution.

and neighbouring functions can influence θ^k . (Note that this is one reason our assumed model independence fails to hold.) Other factors are visual-spatial cues surrounding k . For example, menu length and depth may influence how well users know menu item locations (e.g., longer/deeper menus leave greater possibility for error). Landmarks could accelerate learning, with functions near landmarks—line separators, submenu arrows, disabled (greyed-out) functions, shortcut labels, top or bottom of a (sub)menu—learned more rapidly.

Abstractly, let *Context* denote the set of function usage histories and cues that influence the mental model of k . The learning process can be encoded as $\theta^k = f'(\textit{Context})$. As discussed above, varying contexts afford different learning rates, as illustrated conceptually in Figure 4.8. In terms of the *dynamics* of model strength, we assume that strength M_t^k at time t is a linear combination of the prior strength and strength induced with the new usage context C_t^k :

$$M_t^k = (1 - \lambda)M_{t-1}^k + \lambda C_t^k \quad (4.12)$$

Here, $\lambda \in [0, 1]$ is a learning rate and $C_t^k = f(\textit{Context}_t)$ is the strength associated with $\theta_t^k = f'(\textit{Context}_t)$, but ignores the dynamics of the update process. By incorporating the prior strength, this definition is consistent with the concept of *belief perseverance* [RLH75], whereby people maintain strong beliefs in the face of contrary evidence.

We investigate the relationship between the user’s mental model and several cues and usage histories empirically in Section 4.4.5. From this, we derived $C_t^k = f(\textit{Freq}_t^k, \textit{NB}_t^k)$, where \textit{Freq}^k is the accumulated usage frequency of k and \textit{NB}^k is the accumulated usage frequency of k ’s immediate neighbours. Specifically, we obtained $C_t^k = a^f \log(b^f \textit{Freq}_t^k) + c^f$, with $a^f = 0.11$, $b^f = 0.51$, $c^f = 0.56$ when \textit{NB}_t^k is high, and $a^f = 0.11$, $b^f = 0.44$,



Figure 4.9: Forgetting and changes in the mental model distribution.

$c^f = 0.50$ when NB_t^k is low. We found little evidence of a learning effect with cues (see Section 4.4.5 for details).

4.4.2.3 Forgetting

When functions are not used, we expect users to forget their locations over time. This implies a decrease in model strength (and accuracy in most cases), and an increase in search time. Figure 4.9 illustrates this process, with an unused function becoming more uncertain.

We model the rate of decrease in strength using exponential decay, as suggested by other memory literature [Ebb85]. The impact of disuse on model strength is given by:

$$M_t^k = \beta M_{t-1}^k \quad (4.13)$$

where k is unused at time t , and $0 < \beta < 1$ is the forgetting rate (with respect to model strength, as distinct from decay of model parameters themselves).

Using the processes modeled in Equations (4.12) and (4.13), we tracked model strengths in simulation in the context of menu selection. In a menu with $K = 20$ items and $L = 20$ locations, the user repeatedly selects a series of items according to a Zipf distribution [Zip49]. Figure 4.10 shows the resulting dynamics for six of the menu items. In general, used functions exhibit logarithmic growth in strength while unused functions exhibit exponential decay.

4.4.2.4 Model Disruption

Modeling the effect of adaptive actions on mental models, and strengths, is critical to assessing their utility. When k is moved to a new location, intuitively, we expect θ^k to

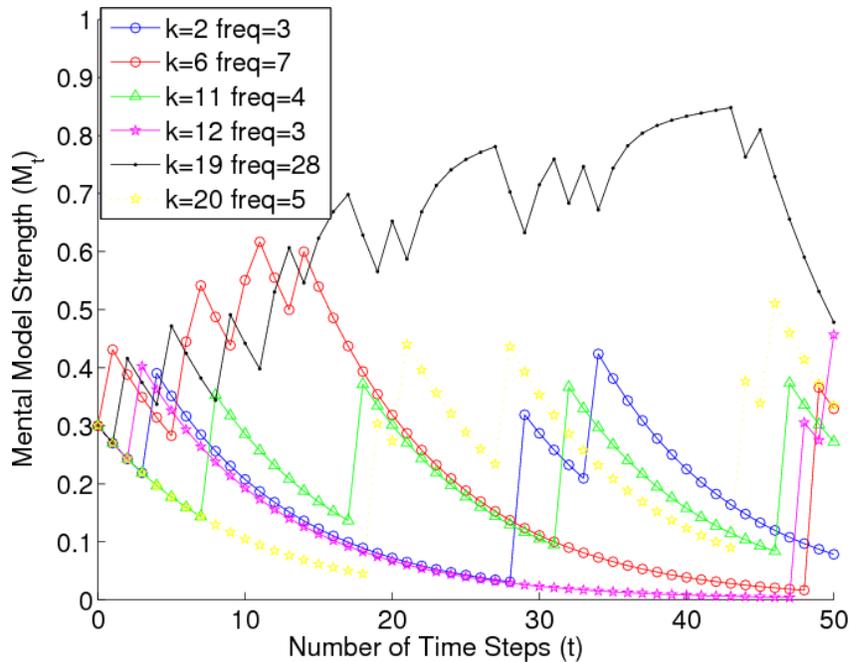


Figure 4.10: Dynamics of strength estimates over 50 time steps with $M_0^k = 0.3$ for all k . Strengths of several functions (indexed as $k = 2, 6$, etc.) with varying usage frequencies (over 50 time steps) are shown.

change in a way that reflects the learning process in the new context (function arrangement) and the previous one. This is due to our expectation that users will retain some memory of k 's previous location, even after realizing it has been moved (an effect like “muscle memory”).

Formally, we capture this effect as a mixture of two models. Let ϕ^k denote a hypothetical model learned (using the learning and forgetting processes above) as if the user had no experience with k prior to the adaptation that moved it. The user's model at time t following the adaptation is given by the mixture $\theta_t^k = (1 - \alpha)\theta_{t-1}^k + \alpha\phi_{t-1}^k$, where α denotes a learning rate (the rate at which the new model replaces the old one). The process is illustrated in Figure 4.11 where k is moved to a new location after $t = 3$. Notice that the “new” model (which is mixed with the old) also evolves over time as the user gains experience with the “new” location.

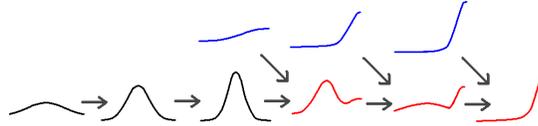


Figure 4.11: Changes in θ^k as k is moved to a new location at time $t = 3$. The bottom distribution represents θ^k and the consequences of moving k . The top distribution represents the hypothetical mental model ϕ^k associated with learning k in a new context. Prior to $t = 3$, the dynamics of θ^k following the regular learning and forgetting processes. Once k is moved, the resulting distribution is $\theta_t^k = (1 - \alpha)\theta_{t-1}^k + \alpha\phi_{t-1}^k$.

In what follows, we model impact of disruption *on model strength* M^k instead of the model θ^k itself. This approximation is more tractable (requiring tracking strength only, not models). We assume that $M_t^k = (1 - \alpha)M_{t-1}^k + \alpha M(\phi_{t-1}^k)$. (The strength learning rate α will not be the same as the model learning rate, but can be derived from it.)

Adaptive actions disrupt a user’s mental model and incur two types of costs. First, referring to Figure 4.11, we see a location change initially has a negative impact on task performance or search behaviour, due to the reduction in model strength and accuracy. At $t = 3$, θ^k is peaked and user can assess k quickly. Immediately after the move, the user will search for k at the old location, not find it, and need to discover the new location (e.g., using visual linear search). This incurs an (objective) *disruption time*. In addition, the user probably experiences some level of frustration (due to both mental effort and increased task time), which is reflected in a subjective *annoyance* factor. In Section 4.4.6, we conduct an experiment to learn the objective disruption time. In practice, the annoyance factor may amplify the disruption time so that users who dislike adaptations perceive a more significant cost. We do not address annoyance further here. Notice, of course, that disruption time decreases over time as the new model is learned. We discuss this further in the next section.

4.4.3 Decision-Theoretic Action Selection

A typical metric for evaluating adaptive systems is the degree to which they reduce user effort in repetitive tasks (e.g., typing common phrases, or selecting frequently used functions such as menu items or toolbar icons). On the other hand, the costs associated with adaptations are often ignored [HGIB08]. Here, we develop a decision-theoretic model that allows benefits (e.g., selection time savings) and costs (e.g., selection time increases due to disruption) to be traded off against one another. Our model is also sequential, reflecting tradeoffs over time.

4.4.3.1 The Benefits of Savings

An adaptive action provides savings for the user if it moves k to a location that is closer to the user's reach. We adopt Fitts' Law [Fit54] and define the *savings* of moving k from l_{t-1}^k to l_t^k as $S(l_{t-1}^k, l_t^k) = fitts(l_{t-1}^k) - fitts(l_t^k)$, where $fitts(d) = a \log_2(d/w + 1) + b$ with w as the width of a target, d as the distance traversed to reach the target, and a and b are empirical constants. This definition requires an application-specific mapping from widget locations to distances, which reflects specific implementation artifacts (e.g., to access a hidden menu item, hovering or clicking double arrows first).

The savings *expected* of any (single) interaction depends on the probability of function usage. Moreover, functions cannot (usually) be moved in isolation (e.g., others may shift). Thus, we define the *joint expected savings* (JES) of a vector of location moves to be:

$$JES(l_{t-1}^{1:K}, l_t^{1:K}) = \sum_{k=1}^K p^k S(l_{t-1}^k, l_t^k) \quad (4.14)$$

where $1:K$ is shorthand notation for K variables, p^k is k 's estimated probability of usage (e.g., based on sample frequency). Since an adaptive action A fully determines $l_t^{1:K}$ given $l_{t-1}^{1:K}$, we also denote Equation (4.14) as $JES(A|l_{t-1}^{1:K})$.

4.4.3.2 The Costs of Disruption

When an adaptive action occurs, we define disruption time (see Section 4.4.2) as the search time required for the function selection task over and above the time required given the prior user model θ^k . Since this is an objective measure, we devised an experiment in Section 4.4.6 to learn this value. With adaptive action A being one of NONE, SWAP, TOP, and HIDE, we define (initial) disruption time as $D_t^k = g(A|M_t^k)$, where g takes the form $a^d M_t^k + b^d$, with empirical constants of $a_n^d = 0, b_n^d = 0$ for NONE, $a_s^d = 1.4, b_s^d = 0.2$ for SWAP, $a_t^d = 2.8, b_t^d = 0$ for TOP, and $a_h^d = 6.0, b_h^d = 2.9$ for HIDE. (These constants were derived from the experimental results in Section 4.4.6.) Using this definition, we define *joint expected disruption* (JED) analogously to JES:

$$JED(A|M_t^{1:K}) = \sum_{k=1}^K p^k g(A|M_t^k) \quad (4.15)$$

Section 4.4.2 presented the dynamics of model strength as a result of disruption. Generally speaking, when function k is moved as a result of adaptive action A , strength becomes weaker. In the system implementation, we capture this with a disruption factor δ , which weakens the model after applying A :

$$M_t^k = \delta M_t^k \quad (4.16)$$

The factor by which strength weakens is defined as a function of disruption time D_t^k . Specifically, given M_t^k , we expect no change in strength when disruption time $D_t^k = 0$ (i.e., A is NONE). In that case, $\delta = 1$. When D_t^k is small, users may not notice the disruption, so we expect strength to only decrease slightly. As D_t^k increases and becomes noticeable, we expect strength to decrease. The greater disruption time, the less the effect on strength because the mental model will, in the worst case, “reset” itself to the weakest point. The specific function we crafted to model this pattern is a Gaussian function defined as $\delta = \exp^{-(D_t^k/c)^2}$, with $c = 3.977$. In effect, there is minimal impact when disruption time is between 0 to 1.5 second. When $D_t^k > 1.5s$, the impact peaks

and eventually stabilizes beyond $D_t^k > 6s$. These boundaries are developed using the empirical results from Section 4.4.6 as guidelines.

4.4.3.3 Sequential Decision Model

The savings and disruption cost models above focus on a single interaction. In repeated use, both savings and disruption costs accrue over multiple interactions. However, disruption cost diminishes over time as the user develops a new mental model as described in Section 4.4.2. The utility of an adaptive action must take into account the sequential nature of the interaction and total net utility over some horizon of interest. For simplicity, we model the long term effects of the current adaptation assuming no future adaptive actions are taken.

Let \mathcal{H} denote the expected number of interactions with an interface. We assume discount factor $0 \leq \gamma \leq 1$, with long-term savings given by: $\sum_{h=1}^{\mathcal{H}} \gamma^h JES(A|I_{t-1}^{1:K})$. Long-term costs are defined similarly, but with a wrinkle. Unlike savings, the long-term effects of disruption decay over time, as the user learns the new model at rate α . With the old model's retention rate as $1 - \alpha$, we discount JED at rate $1 - \alpha$ (in addition to γ): $\sum_{h=1}^{\mathcal{H}} (1 - \alpha)^h \gamma^h JED(A|M_t^{1:K})$.

While one could sum the two expressions above to obtain the estimated utility of an adaptive action, we wish to account for different user preferences. Specifically, we imagine that some users have a strong preference for interfaces that support fast selection (and more generally, task completion) time, even if they are somewhat “disruptive”; others may prefer a more stable interface, even at the expense of increased expected selection time. This is accomplished heuristically by scoring actions using a convex combination of the scores above, with weight $w_s \in [0, 1]$ applied to JES and $w_d = 1 - w_s$ applied to JED . Users with greater w_s are more tolerant of disruption if this induces selection

savings. This gives the *weighted expected reward* score, $WER(A|M_t^{1:K}, l_{t-1}^{1:K}, w_s, \mathcal{H}, \alpha, \gamma)$:

$$\sum_{h=1}^{\mathcal{H}} [w_s \gamma^h JES(A|l_{t-1}^{1:K}) - w_d (1 - \alpha)^h \gamma^h JED(A|M_t^{1:K})] \quad (4.17)$$

To compute the optimal action, the WER system policy is:

$$A^* = \arg \max_A WER(A|M_t^{1:K}, l_{t-1}^{1:K}, w_s, \mathcal{H}, \alpha, \gamma) \quad (4.18)$$

Since we only consider specific kinds of adaptive actions (i.e., NONE, SWAP, TOP, BOTTOM), only a few location changes are possible (specifically, the neighbouring, top, and bottom locations of l^k).

4.4.4 Simulation Experiments

To illustrate the benefits of our model, we conducted a simulation experiment to compare the performance of several system policies in the context of adaptive menus. The objective is to assess the behaviour of these policies and their ability to reason the tradeoffs between potential time savings versus disrupting the user’s mental model. Moreover, since this tradeoff may differ across users, we would like to assess the extent to which these adaptive policies respect user preferences. We report analogous usability results with real users in Section 4.4.7.

The scenario is repeated interaction focusing on menu selection tasks. In each session, a (simulated) user must select an item, drawn from a predefined item distribution. \mathcal{H} items are drawn in turn and the system can adapt its menu according to some policy after each interaction.

4.4.4.1 Comparison Policies

We compare seven menu policies with respect to selection and disruption. BEST STATIC provides an upper-bound on performance: it presents the menu layout with items sorted in descending frequency according to the predefined distribution. Notice that in practice,

a user’s precise usage frequency over functions cannot be known a priori, hence BEST STATIC is “optimistic”. Aside from BEST STATIC, no other policy has access to the item distribution. At the opposite end of the spectrum, we tested the adaptive policy RANDOM- N , which randomly moves N items at each interaction (it is maximally disruptive). The remaining policies make use of *estimated* item distributions to adapt their menus.

Split menus have been shown to offer faster selection performance than various other static and adaptive menus [SS94]. A split menu consists of two areas—the top (adaptive) partition has N menu items with highest estimated usage (in order), while the bottom (static) partition has $K - N$ remaining items arranged in a fixed (default) ordering. Setting $N = 4$ is understood as providing good performance in practice [SS94]. We refer to this policy as SPLIT-4. (Note that refinements of split menu design have been proposed recently, e.g., the “copy” variant [GCH⁺05], which we discuss further below.)

Our adaptive policy is denoted WER(w_s)- N , and is parameterized by the savings weight (and hence, disruption weight) and the number of items moved simultaneously (N). We implement action selection in Equation (4.18) greedily (the best move is derived first, then the next best *given* the first, etc.). We test three versions by varying $w_s = 0.1, 0.5, 0.9$. Finally, we test the JES- N policy, maximizing joint expected savings from Equation (4.14). This alternative adaptive approach ignores disruption (and hence need not reason sequentially).

We investigate these policies under two item distributions: Zipf (which models actual function usage frequencies [GW88]) and uniform. We fix the number of menu items to 20 ($K = 20$ and $L = 20$), the learning rates to $\lambda = 0.5$ and $\alpha = 0.5$, the forgetting rate to $\beta = 0.9$, the discount factor to $\gamma = 0.95$, and the horizon to $\mathcal{H} = 50$. To enable comparative results with SPLIT-4, all the adaptive systems take TOP actions only with $N = 4$. All the systems estimate frequency using normalized sample frequency.

4.4.4.2 Simulated Users

The simulated users select menu items by sampling from the predefined item distribution. To model selection times, we adopt the model of Cockburn et al. [CGG07] which accounts for novice and expert performance (which is necessary in adaptive systems since interface changes can lower a user’s degree of expertise). In this model, item selection requires the user to (i) mentally decide to select an item and (ii) physically select it. Selection time for item k is $T^k = T_d^k + T_p^k$ (i.e., decision time plus pointing time). Pointing time is roughly the same for everyone (using Fitts’ law). However, decision time varies with *expertise* with respect to k ’s location. Decision time is modeled as a linear combination of novice and expert search times with $\epsilon^k \in [0, 1]$ denoting the user’s expertise with k [CGG07]:

$$T_d^k = (1 - \epsilon^k)T_{vs} + \epsilon^k T_{hh}^k \quad (4.19)$$

where T_{vs} and T_{hh} are previously proposed search models for novices and experts respectively (see [CGG07] for details).

Since the literature does not offer a generally accepted model for estimating user expertise in adaptive systems, we equate $\epsilon^k = M_t^k$ based on our mental model strength estimate. In effect, when $M_t^k = 1$ (strong), the user is a complete expert with $\epsilon^k = 1$. As M_t^k decreases (becomes weaker), ϵ^k approaches to the value of a novice.

4.4.4.3 Results

We define *selection time* as the time predicted according to [CGG07], and *disruption time* as any additional search time (Section 4.4.3). To assess the impact of adaptation on learnability, we measure how often a user develops a “strong” mental model of any functions in the session (*total strong models*). We deem a model to be strong when $M_t^k > 0.4$, a threshold based on the sample median of strengths from experiments in Section 4.4.5. We also measure how often a strong mental model is disrupted. We define a *strong move* to be any action that moves a function with a strong model to a

Table 4.5: Results using a Zipf distribution with $K = 20$ menu items, averaged over 100 runs, each with a horizon $\mathcal{H} = 50$. Times in milliseconds.

Method	N	Selection	Disruption	Total	Percent
		Time	Time	Strong	Strong
				Models	Moves
BEST STATIC	0	1197	0	152	0.00
RANDOM	4	1306	196	136	56.80
SPLIT	4	1213	22	150	3.11
JES	4	1301	130	133	36.31
WER(.1)	4	1296	0	152	0.00
WER(.5)	4	1284	6	152	0.18
WER(.9)	4	1273	50	149	2.58

new location, and record the percentage of strong models that are moved (*percentage of strong moves*). Ultimately, we are interested in systems that can tradeoff reduction in selection time (via adaptation) and stability (by minimizing disruption of strong models, where costs are greatest).

The results using a Zipf distribution is presented in Table 4.5. We report the average times based on 100 simulation runs. Note that computation times are fast in all cases ($< 50ms$). BEST STATIC is the gold standard for desirable selection time and stability. SPLIT-4 supports fast selection, as is evident from the literature. All WER variants perform about the same, about 60ms to 83ms slower than SPLIT-4 and 5ms to 28ms faster than JES-4. RANDOM-4 is a lower baseline and induces an undesirable amount of disruption. As expected, JES-4 ignores disruption so it has a cost close to that of RANDOM-4. The remaining methods induce a similar and much smaller range of disruption times. Although these times are small, the associated, subjective annoyance

factor of adaptation should play a role in amplifying overall costs. As a result, we expect task completion times in practice to be greater than the sum of the predicted selection and disruption times (we see this in the usability results below). Thus, the ability to accommodate user preference toward adaptation is crucial.

When the WER weight setting is $w_s = 0.1$ (i.e., representing a user who cares most about minimizing disruption), no disruption is induced. Indeed, our data indicates this policy behaves like a static system so as to not annoy this type of user. Only when setting $w_s = 0.9$ (i.e., a user who prefers to maximize selection performance) is when we see more disruption than SPLIT-4.

The more a system adapts its interface, the less the user is able to learn and develop a strong mental model of it. We see that the number of total strong models that RANDOM-4 offers is much less than that of BEST STATIC. Since WER(.1)-4 behaves like a static system, it also offers just as many opportunities to develop strong mental models as BEST STATIC. We see a similar pattern among these systems when comparing the percentage of strong moves made. When comparing how other systems perform with respect to the percentage of strong moves made, we see that all the WER variants make fewer disturbances than SPLIT-4. Note that having a lower percentage of strong moves does not imply a faster selection time; the policy may move more functions with weaker model strengths. Indeed, we see this pattern in the WER variants in comparison to SPLIT-4 in Table 4.5. This suggests the WER policy reasons that disrupting weaker mental models is better than disrupting strong ones.

Results using a uniform frequency distribution are presented in Table 4.6. Not surprisingly, selection time is difficult to optimize regardless of the policy; users have fewer opportunities to develop strong models (since experience is distributed over more items). As a result, selection times are slower and strong models are rarer. The performance across other dimensions is similar for all the policies, with the notable pattern that all WER variants perform better than SPLIT-4 in all dimensions. Overall, our WER policy

Table 4.6: Results using a uniform distribution with $K = 20$ menu items, averaged over 100 runs, each with a horizon $\mathcal{H} = 50$. Times in milliseconds.

Method	N	Selection	Disruption	Total	Percent
		Time	Time	Strong	Strong
				Models	Moves
BEST STATIC	0	1700	0	100	0.00
RANDOM	4	1701	120	93	49.61
SPLIT	4	1702	44	95	7.33
JES	4	1703	82	91	45.60
WER(.1)	4	1700	0	100	0.00
WER(.5)	4	1700	2	100	0.10
WER(.9)	4	1702	28	97	4.42

is able to tradeoff time savings with associated disruption cost, and is able to adapt its behaviour to user preferences.

4.4.5 Learning Mental Model Dynamics

Our first data collection experiment is designed to determine the kinds of visual-spatial cues and usage histories that define the learning context for model strength. First, participants were faced with a series of controlled menu selection tasks. This allowed participants to build a mental model of our experimental interface. Second, the same participants were presented with a series of recall tasks designed to assess the acquired model strengths. In total, we collected data from forty-eight participants.

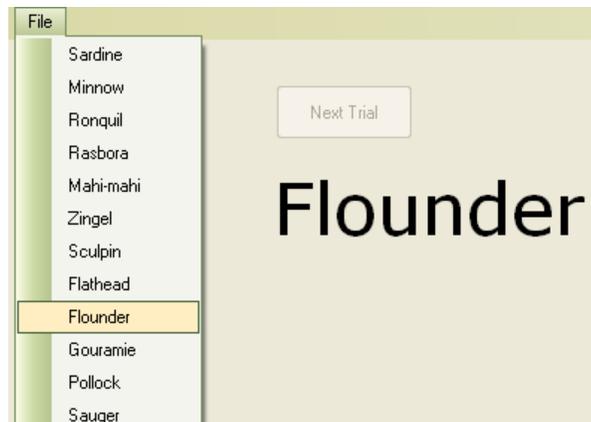


Figure 4.12: Experiment prototype with desktop menu.

4.4.5.1 Training Session

The training session was designed to accommodate a range of cues and usage conditions in order to test the effect of these variables on the mental model. Participants were asked to complete a set of twenty-nine menu selection tasks as accurately and as quickly as possible using abstract labels in a pull-down menu as shown in Figure 4.12. Only four distinct (randomly chosen) menu items were used as targets among the twenty-nine tasks. If a mistake was made, the participant had to redo the task. This set of tasks is designed to mimic a particular *history* of application use, so we can probe the participant’s mental model (in the next session) knowing the usage history that “created” it. In each trial, we measured task completion time and accuracy. We used these results to filter out participants who did not take the experiment seriously.

While we recognize the potential impact of all of the variables discussed in Section 4.4.2, we restricted our attention to two cues and two usage variables:

- *Length*: Total number of menu items (shown and hidden) belonging to the menu. Values: $K=20, 40$.
- *Freq*: Target usage frequency. One usage frequency is assigned to each of the four distinct targets. Values: 1, 4, 8, 16. These give the 29 tasks.

- *Landmark*: A line separator in the menu. Values: none, line above the item with $Freq = 16$, line above the item with $Freq = 1$.
- *NB*: Usage of functions near the target. Values: low (neighbouring items are used zero times), high (a neighbouring item is used 16 times).

As we expect $Freq$ to play a key role in determining model strength, we use a denser set of test values (than prescribed by Zipf). Using four items from a menu with $K = 40$ items mimics real-world scenarios with complicated interfaces where only 10% of the interface is used. Four of $K = 20$ items reflects a denser 20% functionality usage.

4.4.5.2 Recall Session

Once training is completed, participants have developed mental models $\theta^1, \dots, \theta^K$ for each of the K items in the test application (for cases of $K = 20$ and $K = 40$). The recall session is designed to allow estimation of mental model strengths by asking participants to carry out a series of *recall* tasks. Given menu item k , we asked participants to recall its location on a new menu, which has the same interface settings except the menu labels are replaced with *filler* labels. The purpose of these filler labels is to see how well participants remember the actual location of the menu items. Fillers are designed to preserve the length of the original label using a random sequence of consonants. For example, the corresponding filler for “Flounder” is “Xtnxctzr”. To avoid having label length as an added memory cue, we ensured neighbouring labels have roughly the same lengths and instructed participants to complete the tasks as quickly as possible so to not rely on additional cues.

To create the set of target items for this recall session, we first took the four targets from training and added their near-neighbours — i.e., all menu items within a distance of three — to the set. For example, if one of the original four targets was in location l , then items at positions $l + 1, \dots, l + 3$ and $l - 1, \dots, l - 3$ were added to the recall

set as well. This yields a total of 28 ($= 4 \times 7$) items. However, the condition with high frequency neighbours (i.e., $NB = high$) require the target with $Freq = 16$ be located close to the other targets. Therefore, these twenty-eight items are not distinct. As a result, the recall task has between ten to twenty-five unique target items across all the conditions.

Participants were asked to recall each distinct item three times, resulting in thirty to seventy-five recall tasks presented in random order. In each task, participants were asked to identify the location of the targets as accurately and as quickly as possible. For each unique target, the participant responded with three samples of his mental model of that target's location (one per repeated trial), three corresponding response times, and a self-reported confidence score (how well he thinks he knows the original location, on a Likert scale).

4.4.5.3 Experiment Results

We used the task completion times in the training session to identify outliers so that if the completion times did not generally decrease for high frequency items, we assumed the participant did not pay attention. With this criterion, we discarded data from five participants and kept forty-eight for analysis. Next, we present the results from the recall session.

Our first task is to compute the mental model strength. Recall our definition of model strength in terms of entropy is $1 - H(\theta^k)/H_L^+$ for an interface with L locations (here, $L = Length$). To compute strength, we estimated the model distribution using the recalled locations. For each unique target, the three responses were fit to a discrete normal distribution (fit using the sample mean and standard deviation of the three responses). Strength was computed using this distribution. This provided us with a single strength estimate, an average response time, and a confidence score for each of the ten to twenty-five unique targets.

Across all participant-target pairs, the median strength is 0.40. We also measured the correlation between strength, confidence, and response time. We found that strength and confidence are positively correlated ($r = 0.46, p < 0.01$) while response time is not significantly correlated with either strength or confidence ($r = -0.03$ and $r = 0.04$ respectively). This suggests our definition of strength assesses the participants' beliefs of how well they think they know the function locations.

Next, we identify relevant independent variables in modeling the quantitative relationships. Using factorial ANOVA analysis on all the condition variables and following the assumptions of independence, normality, and homogeneity of variances¹, we found that *NB* and *Freq* have a significant effect on strength ($p < 0.01$) and *Length* has only a marginally significant effect ($p < 0.1$). Thus, the variables *NB* and *Freq* provide sufficient context to explain the data, so our strength estimate function is best modeled as $f(\text{Freq}^k, \text{NB}^k)$.

Finally, we fit the data to learn the quantitative relationships of interest. As we expected frequency to have less effect as it increases, we chose to fit the data using a logarithmic function. We fit a separate function for each value of *NB*. Figure 4.13 shows the averaged data and the regression results. As presented in Section 4.4.3, we have $0.11 \log(0.51\text{Freq} + 1) + 0.56$ with $r^2 = 0.9$ when *NB* = *high*, and $0.11 \log(0.44\text{Freq} + 1) + 0.50$ with $r^2 = 0.9$ when *NB* = *low*. This is the function used to define the model strength induced with the new usage context in Section 4.4.2.

4.4.6 Learning the Cost of Disruption

The experiment described in this section attempts to assess the degree of disruption induced by adaptive actions in menu selection. It was conducted as a continuation of the Recall experiment in Section 4.4.5 with the same forty-eight participants.

¹However, these assumptions remain to be tested, ideally with larger population samples. As such, our results provide suggestive evidence only.

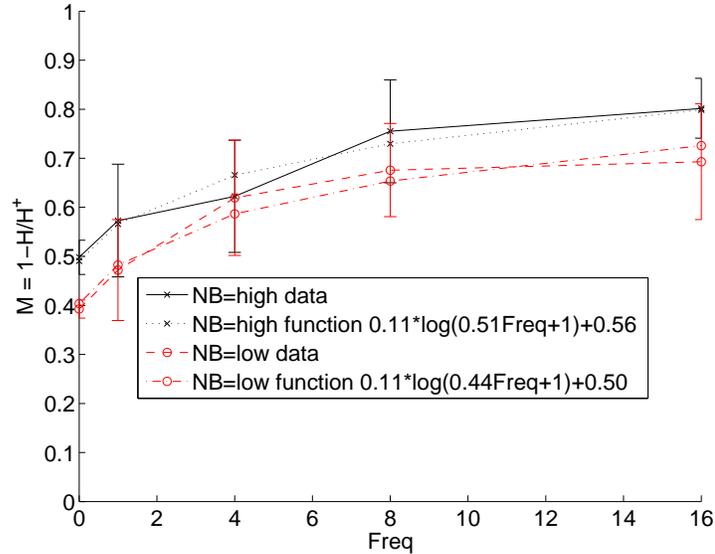


Figure 4.13: Resulting functions for mental model strength.

4.4.6.1 Experimental Set-Up

As defined in Section 4.4.3, the disruption of moving function k is given by $D_t^k = g(A|M_t^k)$. Our aim is to learn this function in the menu selection task, while restricting the action A to changing only one menu item at a time (i.e., $N = 1$). For convenience in this initial study, we treat disruption as additive in what follows, with total disruption being the sum of the disruptions over all functions. (The accuracy of this assumption will need to be verified in future research and experiments.)

We trained the participant’s mental model as described in Section 4.4.5 while keeping track of all task completion times. To induce disruption on the mental model, we applied one of the four adaptive actions (TOP, SWAP, HIDE, NONE) and then asked the participant to select the (potentially) moved target. Thereafter, we asked the participant to indicate whether the target was moved. If no, a self-reported disruption score of 0 was recorded. If yes, we further asked the participant to report the disruptiveness of that adaptation on a five-point Likert scale. Since one of our adaptive actions is to hide menu items, the menu always has 10% of its items hidden. Otherwise, the same interface with

different text labels was used.

For simplicity, we use a subset of the usage frequencies from Section 4.4.5 to create our target items. We randomly chose three target items from the menu and assigned them frequencies one, four, and sixteen respectively, creating a total of twenty-one selection tasks with three distinct targets. These tasks make up the training component in this experiment. We then augmented this set of tasks with adaptive actions and additional selection tasks as follows: after selecting a target p times (where p is the item's associated usage frequency), the system moves its location (as dictated by the chosen action A) and asks the participant to select the (potentially) moved target. This latter component makes up the disruption phase of the experiment. With three target items, this adds three new tasks to the trial². In total, each participant carried out twenty-four menu selection tasks.

Since we are interested in learning the disruption for every combination of usage frequency and system action under each experiment condition, we designed *four* sets of target items and associated selection tasks according to the above procedure. In total, we created twelve distinct target items and ninety-six menu selection tasks. Ideally, a separate experiment would be run for each combination of the condition variables and system action. However, the resulting protocol is too large, and would either be overwhelming for participants in a within-subjects experiment design or logistically infeasible for a between-subjects design. As a compromise, our conditions here vary only in *Length* and *Freq*, and aggregated the other variables into one experiment. This experiment thus takes just an initial step in assessing disruption time.

²Ideally, each of these new tasks should be trained independently. Due to cognitive demands that such a design would impose on the users, we combined the training component of these tasks for simplicity.

4.4.6.2 Results

To estimate disruption time, we subtracted task completion time of the corresponding condition from the training phase from the task time in this new disruption phase. For example, a participant selects item k with $Freq = 16$ in the training phase. Disruption time is the task completion time for selecting k the seventeenth time (after it has potentially moved), minus its 16th task completion time. This gives us a crude assessment of the additional search time induced by the adaptive action. Correlation between disruption time and self-reported disruption scores are positive and significant ($r = 0.40, p < 0.01$). On average, we found the mean disruption time is about 1.5s with a disruption score of 2 (which corresponds to noticing a small amount of disruption), and the mean disruption time is about 6s with a disruption score of 5 (highly disruptive). We used these times to compute δ in Equation (4.16).

Since this experiment is conducted following the Recall experiment, we took the estimated strength values from the same participant’s corresponding conditions and used them in fitting D_t^k . The data was noisy in general, so we binned the strength estimates into three equally-sized buckets and analyzed the disruption times with respect to a weak, medium-strength, and strong mental model. The mean values for these bins are 0.26, 0.66, and 0.90 respectively.

In general, we expect disruption time to increase as strength increases. Using the empirical disruption times for $A = \text{NONE}$ as a baseline, we chose to fit the data using linear regression for simplicity. Figure 4.14 shows the averaged data and the following regression results: when $A = \text{Swap}$, we have $D = 1423M + 158$ with $r^2 = 0.5$, when $A = \text{Top}$, we have $D = 2865M$ with $r^2 = 0.8$, and when $A = \text{Hide}$, we have $D = 6033M + 2901$ with $r^2 = 0.9$. Overall, we see that disruption time is positively correlated with model strength, and most significantly so with HIDE. Note that existing predictive pointing models (e.g., Fitts law) do not account for a user’s mental model, and at best attempt to reflect only a user’s expertise level (e.g., [CGG07]), neither of which adequately accounts

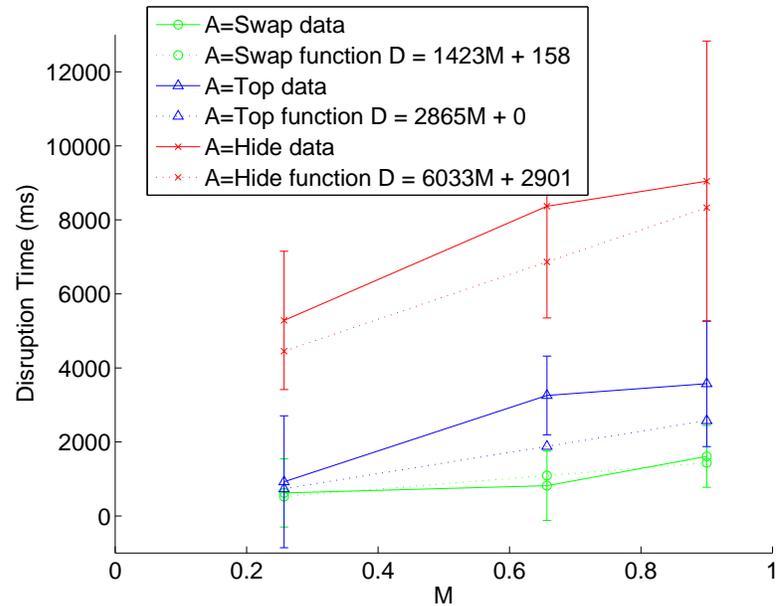


Figure 4.14: Resulting functions for disruption time.

for added disruption time. Our results suggest the need for such a model in adaptive systems.

4.4.7 Usability Experiment

We concluded this study with a usability experiment designed to test and verify the simulation results from Section 4.4.4 with real users, adopting model parameters estimated via the preceding experiments. We adopt the same set-up and evaluation metrics as those in the simulation experiments from Section 4.4.4. In total, we collected data from eight participants.

4.4.7.1 Experimental Set-Up

Following the experiment in Section 4.4.4, we created menu selection tasks with these systems: BEST STATIC, RANDOM-4, SPLIT-4, and WER(w_s)-4. We chose BEST STATIC and RANDOM-4 as they provide baseline results, and SPLIT-4 as a plausible competitor to

our model. Another competing approach not investigated here is the copy variant of split menus [GCH⁺05]—items are copied rather than moved to more convenient locations—which has been shown to be preferred by users. To offer a fair comparison, we could augment our adaptive WER(w_s)-4 policy with a COPY action. We leave this possibility to future research. To create the same visual menus in each system, the line typically separating the top and bottom partitions in split menus is removed. All parameter values used in this study are identical to those in Section 4.4.4.

There are two parts to this experiment. The first is a within-subjects experiment which asked participants to compare the 4 systems by carrying out 50 menu selection tasks with targets sampled from a Zipf frequency distribution. To help differentiate the experience, each system was designed with a different set of menu labels (e.g., fish, colors, fruits, animals). The second part follows the same design except it uses a uniform distribution rather than Zipf. In all cases, we used an interface similar to the one shown in Figure 4.12 and fixed menu length to 20. The presentation order of the four systems and the two parts were counter-balanced across participants. Since different menu labels and selection targets were used in each condition, we expect learning effect to be minimal.

We let participants explore the interface using RANDOM-4 until they were comfortable. To determine their preference toward adaptive systems, we asked a multiple choice question, “Would you use adaptive menus if they were designed to SPEED UP the tasks?” To a response of “yes”, we assigned the weight setting $w_s = .9$ in our WER system (with $w_d = 1 - w_s$), denoting the participant has a strong preference to maximize savings at the expense of added disruption. On the other hand, a response of “no” was assigned $w_s = .1$, denoting the participant has a strong preference to minimize disruption. Finally, a response of “maybe” was assigned $w_s = .5$. At the end of each part of the experiment, we asked participants to rate each system on a five-point Likert scale based on frustration, ease of use, and efficiency.

4.4.7.2 Results

In each trial, we logged the task completion time (as opposed to the predicted selection time in the simulation evaluation) and estimated the corresponding disruption time. Following the format from the simulation, we report the objective usability results from the Zipf condition in Table 4.7. Among the eight participants, three used the weight setting of $w_s = .9$ for our WER system, four used $w_s = .5$, and one used $w_s = .1$. Since we do not have an equal number of participants for each weight setting, we also aggregated their results together to provide an overall performance on WER.

In contrast to the predicted selection times from the simulation results, the *measured* task times for the three adaptive systems are much higher. We suspect this effect is due to the participant’s subjective annoyance factor toward the system’s adaptations which resulted in an increased overhead.

Overall, we are interested in comparing the performance of using WER to that of using SPLIT. In general, we see similar results as those in the simulation: WER is competitive with SPLIT-4 when comparing task and disruption times. Unlike the simulation, WER(.9)-4, whose goal is to maximize savings, does better than SPLIT-4 on all dimensions. The results show WER(.1)-4, whose goal is to minimize disruption, offers more opportunities for learning strong models than SPLIT-4. Similar to the simulation results, the WER policy has a lower percentage of strong moves but (generally) a higher task time, suggesting that the WER prefers to move functions with weaker mental models when adaptation is necessary.

The advantages of our method are made more obvious when the tasks are created from a uniform distribution. Table 4.8 shows these usability results. In comparison to SPLIT-4, we see that WER(.5)-4 is faster in task time and WER(.1)-4 offers more opportunities to developing strong mental models.

Lastly, we report the post-questionnaire results in Figure 4.15. Although no significance was found, in large part due to having to divide the number of participants into

Table 4.7: Usability results using a Zipf distribution. Times in milliseconds.

Method	N	Estimated		Total	Percent
		Task Time	Disrupt. Time	Strong Models	Strong Moves
BEST STATIC	0	1513	0	134	0.0
RANDOM	4	2966	779	82	59.9
SPLIT	4	1760	26	111	9.8
WER(<i>all</i>)	4	1817	21	121	5.2
WER(.1)	4	2123	24	135	3.7
WER(.5)	4	1864	23	118	5.1
WER(.9)	4	1651	17	119	5.8

Table 4.8: Usability results using a uniform distribution. Times in milliseconds.

Method	N	Estimated		Total	Percent
		Task Time	Disrupt. Time	Strong Models	Strong Moves
BEST STATIC	0	2335	0	82	0.0
RANDOM	4	3322	993	54	50.9
SPLIT	4	3311	75	56	25.6
WER(<i>all</i>)	4	2913	47	60	29.7
WER(.1)	4	3546	53	84	1.2
WER(.5)	4	2792	27	63	33.5
WER(.9)	4	2865	31	49	34.2

three WER weight cases, we see that on average, participants reported that our WER-4 system is less frustrating, easier to use, and more efficient than SPLIT-4. Overall,

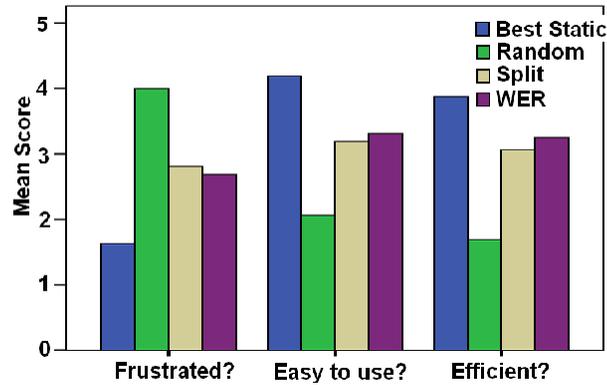


Figure 4.15: Subjective results.

our usability results indicate that the various WER policies provide a reasonable range of adaptivity and the pattern of results confirms and amplifies the conclusions in the simulation experiment.

4.5 Summary

In this chapter, we described a set of user characteristics that play a role in explaining individual preferences in intelligent software adaptation systems. These characteristics include general user trait variables — such as distractibility, independence, speed, tolerance (to bloat) — as well as variables describing the user’s mental model and attitude toward the system or the task — such as frustration, neediness, concentration, and mental model strength. In particular, we conducted two case studies that demonstrated the use of a selected subset of user characteristics in different applications.

In Section 4.3, we presented an adaptive text editor that suggests word completion help based on a myopic decision-theoretic policy. The focus of this case study was the development of a Bayesian user model that inferred the user’s frustration, neediness, independence, and distractibility. The ability to model these characteristics enables the system to adapt its interactivity level to suit the preferences of the user.

The second study reported in Section 4.4 presented an adaptive menu that changes the location of certain menu items based on the inferred strength of the user’s mental model of menu item locations, the associated (long term) disruption cost in changing those locations, and the potential (long term) savings involved. The focus of this case study was the development of a probabilistic mental model, the design of data collection experiments in learning various model parameters, and the value of using this information in an intelligent system’s decision making process. While some of the choices were heuristic, the system demonstrated higher value in comparison to alternatives that did not model the impact that adaptive actions have on the user’s mental model.

Overall, we identified a set of user characteristics to guide the design and development of intelligent software adaptation systems and demonstrated the development of the User Characteristics Component within the DAISI framework through two case studies. The empirical evidence in these studies suggests that there is value in modeling user characteristics to adapt to users’ changing states, and that users perceive the kind of suggestion utility defined in our Action Selection Component.

Chapter 5

Personalized Goal Recognition

The second major component in our DAISI framework is the User Goal Component used for predicting user goals. One way to approach goal recognition is to focus on a set of domain-specific goals and predict the current user goal by developing models and techniques that differentiate these goals from each other. We adopt this domain-specific approach, but also tailor the techniques to recognize individual patterns in achieving those goals. In particular, we develop a goal recognition component that passively learns user-specific goals in the domain of slide presentations, using PowerPoint 2003 as the testbed application. Although the details of our goal recognition model are application-specific, each section in this chapter outlines the steps required for building a general recognition model that can be applied to other applications besides PowerPoint. In particular, so long as we have access to the relevant event logs, user goals can be automatically learned and our User Goal Component can be applied to recognize those goals, irrespective of the application setting. The overall design for the goal recognition component is illustrated in Figure 5.1. Here, we introduce the general terminology and methodology used in the overall goal model and describe the architecture.

Following the DAISI framework, user interaction is partitioned into sequences of *episodes*, where we assume there is one intended goal in each episode. To begin, we view

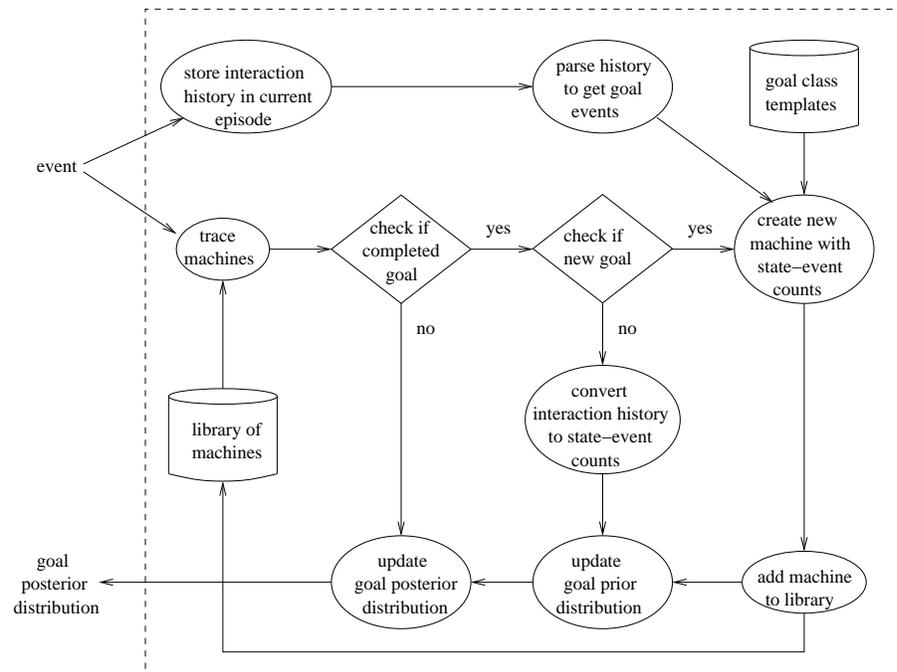


Figure 5.1: A logical diagram of the goal recognition component. The data flow shows the flow of information given a single user event as input and the posterior goal distribution as output in one time step. Ovals denote processes, diamonds denote choice points, and cylinders denote collections of models.

a *user goal* as a target application state intended by the user which can be achieved by executing a sequence of events. We refer to a group of similar user goals that can be concisely defined by specific parameters as a *goal class*. For example, users may want to emphasize a phrase in a slide presentation by changing its font style to red and bold. We consider this to be a *highlighting goal*. The group of highlighting goals that vary by a set of font style parameters is referred to as the *highlighting goal class*. In order to help users execute goals more quickly and/or with less effort, the system needs to learn and store observed goals in a *library*, so that the system may offer help during the execution of those goals in the future. A key to a good recognition model is early recognition of the user’s intended goal. This implies that goals that require the execution of longer event sequences (i) provide more opportunities for the system to suggest the right help, and (ii)

provide help that saves the user more effort that would otherwise have been needed to accomplish these goals manually (i.e., these help actions have higher utility). Therefore, in designing intelligent help systems, we are generally interested in modeling repetitive goals involving reasonably long event sequences. To illustrate the kinds of user goals we are interested in, Section 5.1 develops several goal classes using PowerPoint examples. Predefining a set of goal classes of interest to the recognition component enables our system to focus on a closed set of goals rather than attempting to recognize arbitrary user goals.

In order for the system to learn, store, and predict user goals, we formalize goals using deterministic finite automata (DFAs) which is presented in Section 5.2. This formalism uses the observation set in the POMDP-DAISI from Chapter 3 to define an *event vocabulary* as input symbols to the DFA. Generally speaking, DFA states represent application states corresponding to partial completion of a goal, so that executing events for a user goal corresponds to transitioning in the automata, and achieving the user goal corresponds to accepting the event sequence. To demonstrate, we focus on the highlighting goal class elaborated in Section 5.1 and develop automata for recognizing goals in that class. First, we develop generic *goal templates* that specify the syntax of plausible highlighting goals. These templates are general enough so to encompass variations of goal completion patterns across users and across different usage situations for the same user. Personalization is achieved through a set of *goal machines* that are dynamically constructed from goal templates based on observed user events in the episode. Then, these goal machines are incorporated in the library. While this model is designed specifically for the highlighting goal class for demonstration purposes here, general goal templates and machines can be learned automatically given event data containing goal execution sequences (e.g., [GP00]). To enable the intelligent system to reason with user goals probabilistically, each goal machine is equipped with a stochastic component that models the probability of the user executing an event in each automaton state. As different

event patterns may be observed depending on the circumstances or the user's interaction style, the set of goal machines and the probabilistic information will vary but the set of templates in the system remain unchanged.

During the course of any user interaction, the system traces all the goal machines stored in the library using incoming user events as indicated in the data flow diagram in Figure 5.1. When one of the machines in the library accepts a recognized user goal or when the system observes a special event that designates the start of an episode, the system terminates the current episode and begins a new one. The interaction history observed during that episode is used to update the system's knowledge about the user's goal execution behaviour by updating the stochastic component of the goal machines.

The set of goal machines stored in the library defines the plausible user goals known to the system. We use this information to probabilistically reason about the user's intended goal at any point in time. Recall from Section 3.1.3 that the system maintains a goal distribution within an episode as well as across episodes. In this regard, we define an *episode prior distribution* over the set of plausible goals. This distribution represents the probability of the true goal given no observation. Over time, as completed goals are observed, we update this distribution accordingly. Within each episode, we define an *event prior distribution* that represents the probability of the true goal before an event is observed, and an *event posterior distribution* that represents the probability of the true goal given an observed event sequence. While the episode prior is updated as new goals are observed at the end of each episode, the event prior and event posterior distributions are updated with each observed event within episodes. In effect, the goals with non-zero probability in the posterior distribution are those whose events are consistent with the observed event sequence. Although goal prediction in general should be conditioned on situational context (such as the time of day and the user's focus of attention), we restrict our attention to the application state only for simplicity. The inference procedure and how the goal distribution is updated are presented in Section 5.3.

The output of this procedure is the updated goal distributions.

To evaluate the effectiveness of this goal recognition component, Section 5.4 presents simulation results that illustrates the online computational requirements, accuracy, and utility of the goal model. As a result, we see that the procedure for incremental goal inference is fast. However, exact inference on the user characteristics model that incorporates DFA states becomes slow quite quickly as the number of goals (and thus, machine states) increase. In terms of accuracy, both goal inference and characteristics inference demonstrate good results. Compared to the policy of never helping the user and adaptive policies derived based on the use of a frequency distribution instead of a goal distribution, the policies based on the goal model yield better results with respect to task completion effort, percentage of accepted suggestions, and estimated utility. To confirm these simulation results, we evaluated the goal model in a small user study in Section 5.5 by asking users to carry out a series of highlighting goals in PowerPoint 2003. In comparison to the policy of never helping the user and always helping the user based on a frequency distribution, we confirmed that the use of the goal model provides better utility, both objectively and subjectively as perceived by users. Furthermore, unlike frequency policies that only update their suggestions at the end of each episode, we found that the sequential nature of the highlighting goals calls for adaptive policies that are able to incrementally update suggestions based on each observed user event.

5.1 Classes of Goals

In Chapter 4, we demonstrated the user characteristics model in a text editor application with word prediction suggestions and a menu application with adaptive layout suggestions. In the text editor case, a bigram model was used as a goal model to predict the current word. In the adaptive menu case, a frequency distribution was used as the goal model to predict the current menu item to be selected. Here, we focus on goals

of a more sequential nature, set in a realistic application with an elaborate interface. In particular, this section describes several classes of goals at a descriptive level in the context of PowerPoint 2003. The purpose of this section is to illustrate the variety of potential types of goals that intelligent systems can help users with, so that users may interact with software more efficiently and effectively. The goal classes described here are discovered by analyzing published online academic presentations made in PowerPoint and informal discussions of anecdotal experiences with PowerPoint users. We report on the most commonly occurring goal classes in this section by providing example goal instances, defining the event sequence required to execute such instances, and suggesting a way that intelligent systems could help the user to achieve goals more easily. We elaborate on one of these goal classes — the highlighting goal class — with detailed examples and use it to develop the models and procedures in the rest of this chapter. While we do not elaborate on the modeling details of the other goal classes discussed, the models and inference technique we present for the highlighting goal class below can be adapted and generalized to these classes of goals as well.

5.1.1 Highlighting

The highlighting goal class refers to situations when users want to change the font attributes of a string so it appears different from the default font attributes of other surrounding text. Such strings are referred to as the *target string*, which may be of varying length, consisting of letters, words, or lines. For example, a user may highlight a word to emphasize domain terminology as in the pink words in Figure 5.2(a) by first selecting the target string and then executing a sequence of events to change the string’s font attributes. The resulting pattern used to make these strings stand out from the surrounding text is referred to as the *target pattern*. As well, a user may give emphasis to longer strings at the bottom of a slide to create a “punchline” effect as in Figure 5.2(b). Highlighting patterns may include stylization combinations that change certain aspects

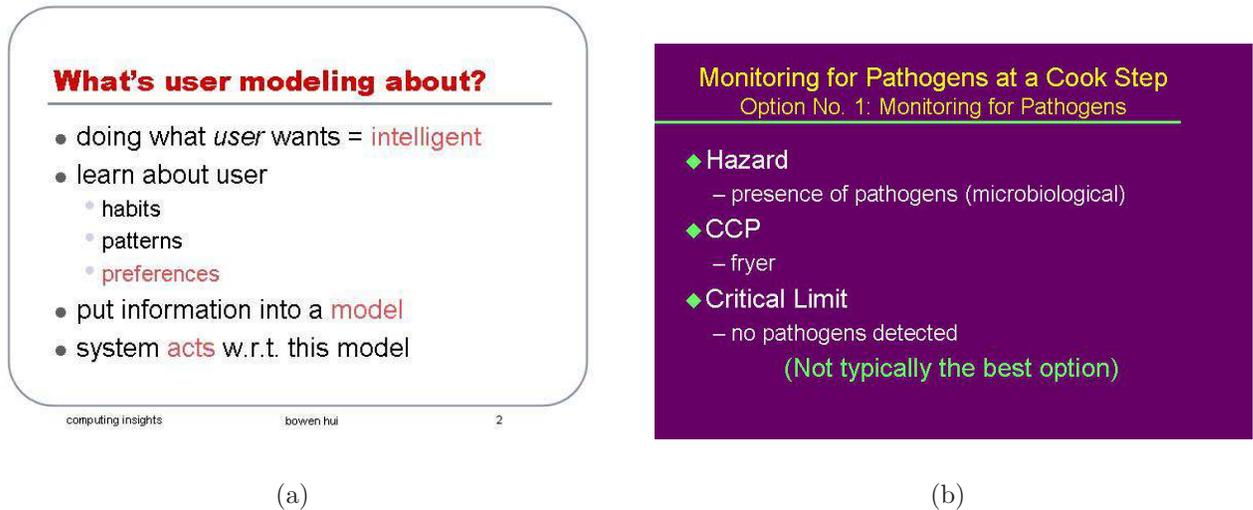


Figure 5.2: Highlighting goals with potential event sequence of (a) selecting the target string and changing its font colour, and (b) selecting the last line as the punchline, centering it, and colouring it differently.

of the selected string, such as font attributes (e.g., font colour, font size, font family) and the string’s immediate paragraph attributes (e.g., line justification, background colour). Once highlighting goals are detected, the system may auto-complete the intended target pattern on the user’s behalf, without the user having to execute each individual font attribute change manually.

Example Event Sequences

To illustrate the kinds of event sequences considered for achieving goals in the highlighting goal class, we provide several examples here. For simplicity, we use more descriptive events in our examples in this section, but will define our event language formally in Section 5.2.2.

Consider a simple example where the user has typed in some text in the slide and wants to highlight a (plain, default styled) string by selecting it and then applying some font attribute changes:

```
select(string), apply_f1, apply_f2, apply_f3
```

In this case, the target pattern consists of font attributes `f1`, `f2`, `f3`. Executing these font attributes in a different order, such as those presented in the next examples, also result in the same target pattern:

```
select(string), apply_f2, apply_f1, apply_f3
```

```
select(string), apply_f3, apply_f1, apply_f2
```

The importance of distinguishing the order of event execution is that it may reflect user preferences. Consider the goal, $g1$, with the target pattern `f1` and `f2`, and a second goal, $g2$, with target pattern `f1`, `f2`, and `f3`. One user may prefer to carry out $g1$ by first executing `f1` then `f2` (say, 90% of the time), while in the context of carrying out $g2$, the user generally prefers to execute the events in the order of `f2`, `f1`, `f3` (perhaps because `f1` and `f3` are located near each other so the user prefers to execute them together). This ordering information provides clues about the user's true intent. In particular, after the system observes that `f1` has been applied to a (plain, default style) string, the system's goal distribution suggests that $g1$ is a more likely goal than $g2$ even though they both have the observed font attribute in common. Although we do not consider scenarios of multiple users, modeling the order of event execution also increases the system's ability for multi-user identification under the assumption that different users prefer to execute the same goals in different ways.

We may also model the user changing his mind during the course of executing these events by undoing a font attribute, or model the application of additional attributes by browsing or pausing as illustrated in the following example:

```
select(string), apply_f1, undo_f1, apply_f2, browse, pause, apply_f3
```

In this case, the target pattern consists of `f2` and `f3`. Other indications that a user has changed his mind include selecting a string with an existing font pattern and changing it as follows:

```
select(string with f1,f2,f3), apply_f4  
select(string with f1,f2,f3), undo_f2, pause, apply_f4, apply_f5
```

Since an automated system can offer suggestions for highlighting completions, the user may also execute events in response the automated help. In particular, the user may accept certain combinations of font attributes as illustrated by the following examples:

```
select(string), apply_f1, accept(f1,f2,f3,f4)  
select(string), accept(f1,f2,f3,f4)  
select(string), apply_f1, accept(f2,f3)  
select(string with f1), apply_f2, apply_f3, accept(f1,f3,f4,f5)
```

After accepting an automated suggestion, the user may further refine the pattern by applying additional font attributes and/or undoing part of the suggestion. These scenarios are illustrated in the following event sequences:

```
select(string), apply_f1, accept(f2,f3), apply_f4  
select(string), accept(f1,f2,f3,f4), undo_f4, undo_f3, apply_f5
```

Although these examples appear simplistic, most approaches to goal recognition assume users do not make mistakes. Thus, the majority of the examples above cannot be handled by existing approaches. Furthermore, most approaches do not distinguish user-specific goals or user-specific patterns of execution. In Section 5.2, we present the formal model for recognizing generic goal classes and personalized goals.

5.1.2 Simple Mathematical References

This section describes a special case of the highlighting goal class that is applied to simple mathematical references. When authoring scientific slides, users often need to introduce mathematical functions and refer to them in later slides. Here, we focus on this particular set of user goals. Specifically, we do not attempt to address scenarios with

U as a Probability Density Function

- Probability density over utility space
 - Explicit repn of our uncertainty about A's utility function
- View **subutilities** **u** as a continuous valued random vector
- Use pdf, $p(\mathbf{u})$, as our prior belief representation
- observations of A's actions as evidence
- Condition evidence on prior to get posterior, $q(\mathbf{u})$
- Reason A's action using $q(\mathbf{u})$
- Problems:
 - $q(\mathbf{u})$ is too complex
 - Hard to compute volume of polytope
- Solution: sample from $q(\mathbf{u})$

April 23rd, 2003 CoGS seminar CKO 2001. Bowen Hu 9

Figure 5.3: Example goals for simple mathematical references that have nested highlighting are “ $q(\mathbf{u})$ ” and “ $p(\mathbf{u})$ ”, where the user would italicize the entire string and boldface the substring “ \mathbf{u} ”. Note that on line 3 of this slide, the string “ \mathbf{u} ” is boldfaced, but since it does not meet the nested highlighting criteria, this goal does not belong to the class of simple mathematical references.

complex equations or slides full of such mathematic notation, because users will be more likely to use equation editors in those cases. The kinds of mathematical references we are interested in are illustrated in Figure 5.3.

The key criteria that differentiate goals in this class from those in the highlighting class are as follows:

1. goals in this class are applied to target strings that only contain a single word
2. goals in this class require *nested highlighting* applied to the same target string

The first criterion states that goals in this class are only applicable to a string of consecutive letters (i.e., not phrases). Because we are interested in references to mathematical functions, we restrict our attention to simple cases of single words only. The second criterion states that nested highlighting is required on the same target string. In other words, the target string may be of a particular font style, but a substring has at least

Table 5.1: Additional examples of simple mathematical references goals.

Goal	Target String	Example Event Sequence
U^*	U*	select(string), italics, select(*), superscript
U_C	UC	select(string), italics, select(C), subscript
V_N^B	VBN	select(string), italics, select(B), superscript, select(N), subscript
$V_n[\mathbf{u}]$	Vn[u]	select(string), italics, select(n), subscript, select(u), bold
$f(\mathbf{x}, \underline{y}, z)$	f(x,y,z)	select(string), italics, select(x,y), bold, select(y,z), underline

one additional font style applied to it. Table 5.1 illustrates additional examples of goals in this class, where *goal* indicates the stylized target pattern desired, *target string* is the set of consecutive letters typed to achieve the goal, and *example event sequence* shows one sequence that achieves the target pattern of the goal.

These examples in Table 5.3 illustrate that multiple font styles are applied to the same target string to create a single mathematical reference. In some cases, non-overlapping substrings of the target string can have different font styles in the goal (e.g., after highlighting the entire string, the substrings “B” and “N” are highlighting separately in the third example). Alternatively, overlapping substrings may also have different font styles (e.g., in the last example, after highlighting the entire string, the substrings “x,y” and “y,z” are highlighted as well).

Since these goals deal with references, the actual highlighted string does not change when repeated in future slides. In other words, if “ $p(\mathbf{u})$ ” is a simple mathematical reference, whenever the user types $p(u)$ and selects this string, we can infer that the user is about to highlight it to match the pre-existing mathematical reference font style.

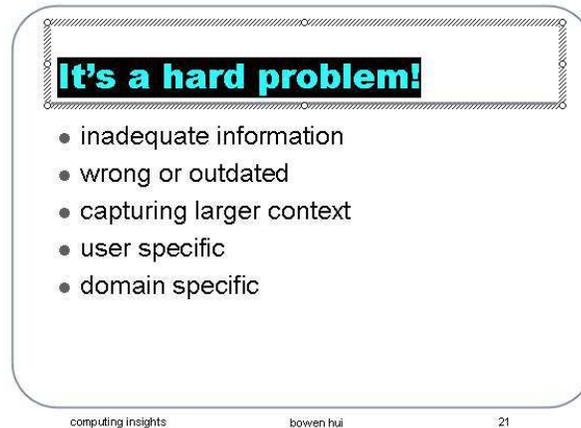


Figure 5.4: An example of changing the title font style on one slide, with the overall goal of changing the titles on all existing slides in the presentation.

Thus, after the user selects a target string, the system can search through a list of stored mathematical references. If a match is found, then the system can suggest a highlighting pattern for the selection.

5.1.3 Edit All Titles

Another special case of the highlighting goal class captures the editing of slide titles. In authoring a new slide presentation, users need to create a blank slide using a default presentation design. This default design specifies the background image used on each slide, as well as the font family and font size used in the title and body of the slides. In cases where users prefer a different font style in the titles, users would need to change all the titles on every existing slide in the presentation to a target pattern. We refer to this sequence of events as the events needed to achieve the goal of editing all titles. An example of this goal is shown in Figure 5.4, where the user selects the entire title as the target string, and changes the font style to a new target pattern.

There are two aspects of this goal class that make it different from the highlighting goal class. First, users may interleave events irrelevant to highlighting with the editing

tasks on each slide. Consider the following scenario. A user has created a presentation with five slides and decides to change the appearance of the titles to a new font style. With this target pattern in mind, he edits the font style of the title on slides 1, 2, and 3. While he is on slide 3, he notices a typographical error which he fixes by erasing a couple of letters. He continues onto slide 4 and 5 to edit those title styles, at which point, the goal of editing all titles is completed.

This scenario illustrates that the goal of editing all the titles may be “interrupted” by events irrelevant to highlighting (i.e., the events of erasing two letters) and “resumed” until all the slides have their titles changed to the target pattern. We refer to this property as *hierarchical*, such that the goal class of edit all titles is a hierarchical version of the highlighting goal class.

A second feature that differentiates goals in this class from those in the highlighting class is that goals here apply not only to the existing slides, but also to the future slides. Continuing with the above scenario, after the user is done with the first five slides, a new blank slide is created for editing. This sixth slide has the default properties of the presentation, so that the title font style is the default pattern, and not the target pattern used for slides 1 to 5. Therefore, in addition to recognizing that the titles in the existing slides need to be changed to the user’s target pattern, the *default* properties of the title style also need to be changed. Once an edit all titles goal has been detected, an intelligent system could change the title styles on all the slides as well as the default title attributes to the target pattern.

5.1.4 Initial Indentation

In some cases, users may want to create points without bullets, such as the examples shown in Figures 5.5. In Figure 5.5(a), we see that among the five points in the slide, only the first one shows the default bullet. One might consider that the string “preferences” on the second line of this slide appears without a bullet because it is too long to fit on the

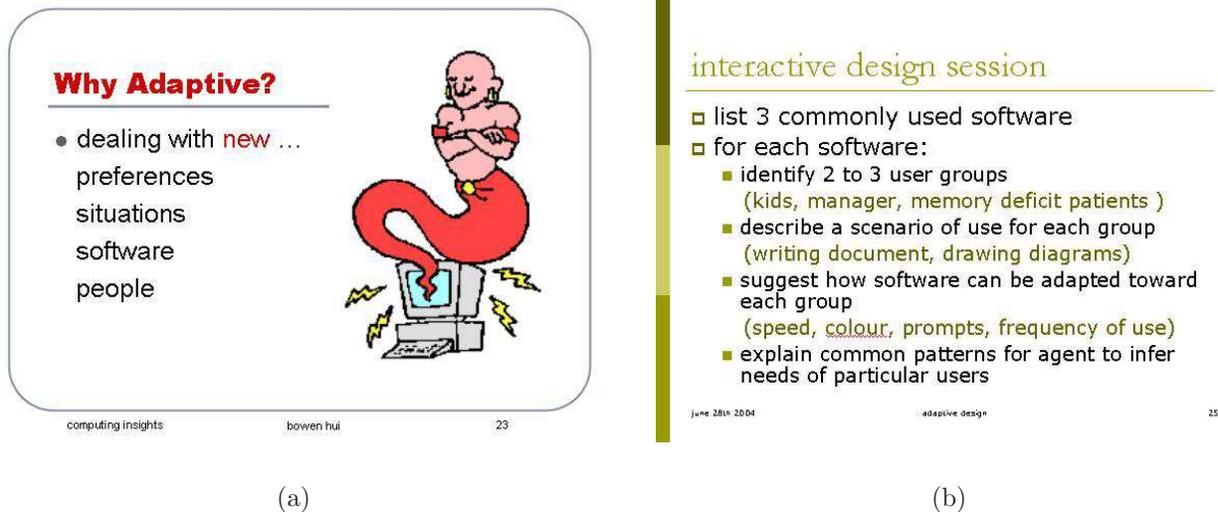


Figure 5.5: Examples of the indentation goal class with (a) consecutive points are manually indented to align with the initial spacing of the previous point, and (b) non-consecutive points are manually indented to align with the initial spacing of the previous points respectively.

previous line. However, by checking the size of the text box on the slide or by checking the spacing in the source presentation file, we know that this was not the case. Thus, in order to display the remaining points in the way shown on the slide in Figure 5.5(a), the user would have had to delete the default bullet and manually align the string of the subsequent points using multiple spaces so the initial spacing matches that of the first point. Similarly in Figure 5.5(b), in order to display the points in brackets without the bullet and have their initial spacing align to that of the line above, the user would have had to delete the default bullet and manually indent the string. Within the event sequence of deleting the bullet and entering spaces, the system could automatically align the initial space on the user’s behalf. We refer to these goals as indentation goals.

5.1.5 Re-Alignment

Aside from text entry, users may also draw diagrams while authoring slide presentations. Objects that can be used in these diagrams include images, text boxes, and shapes.

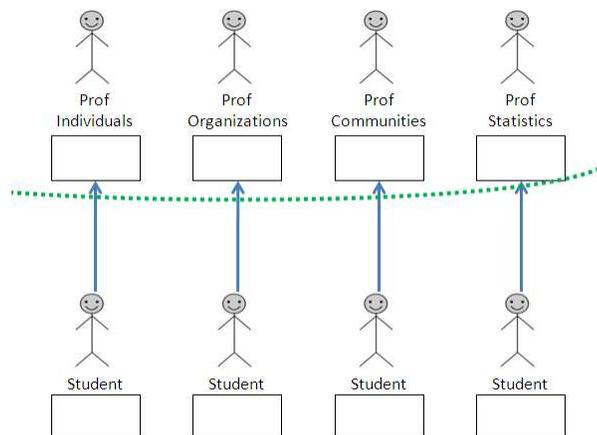


Figure 5.6: Examples of the re-alignment goal class, where the objects in this figure are the stick figure image, the text label, and an empty box. Whenever the size of one of these objects changes (e.g., the string in the text labels), all the objects need to be center-realigned.

Figure 5.6 shows an example of a diagram which consists of stick figure people with text labels and boxes pointing to other similar combination of objects. Each combination of a stick figure, its text label, and its empty box in this figure are center-aligned. In editing the slide presentation, the user may need to change the text labels in the diagram. For example, if the label “Prof Organizations” changed to “Prof Business”, the width of that text label will change, and the associated stick figure and empty box will need to be center-aligned again. An intelligent system may automatically re-align these objects on the user’s behalf. Note that the order of these events is important, because center re-alignment of a combination of objects follows after the resizing of one of these objects. Thus, each time the size of an object changes, its needs to be re-aligned with other associated objects. To accomplish this, the user needs to select the resized object and select any other object that needs to be center-aligned, search for the center alignment function and select it.

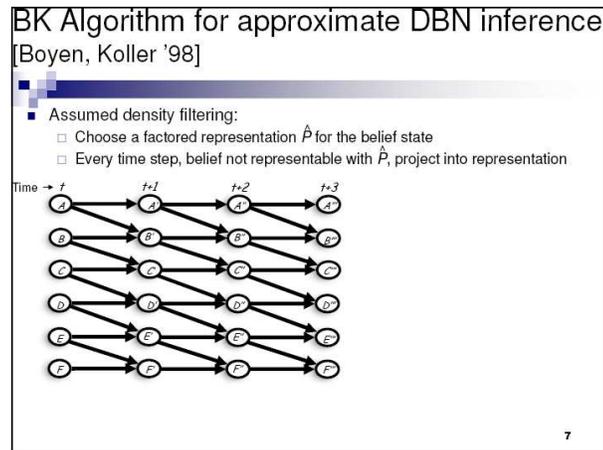


Figure 5.7: Examples of the goal class for connecting nodes and arrows repeatedly.

5.1.6 Multiple Node-Arrow Connection

Another common goal class relevant to editing diagrams is connecting multiple pairs of nodes and arrows. When users create graphs, they need to make multiple copies of nodes and connect them with arrows. An example of a slide with a graph diagram is shown in Figure 5.7. One way to create such a graph is to draw a single node and a single arrow, then make many copies of the original node and many copies of the original arrow, and finally connect them as needed. Let us define a node-arrow connection goal as the user event of dragging an arrow to a node in a diagram. Recognizing consecutive instances of this goal leads to the more general goal of multiple node-arrow connection. Thus, multiple node-arrow connection goals are hierarchical versions of node-arrow connection goals. Graphs with regular and repetitive structure, such as the one in Figure 5.7, are examples of this goal class. The ability for a system to recognize multiple node-arrow connection goals and automatically connect the remaining node-arrow pairs can save the user a lot of manual effort.

5.1.7 Typing Words

The final goal class we discuss is the one that is relevant to all presentations — typing. For simplicity, we view each word as a user goal (rather than every two-word or three-word combinations). As demonstrated in the text editor example from Section 4.3, the kind of automated help that can be developed is the suggestion of a list of potential words to help the user complete the target word without typing the rest of the letters. Consider typical event sequences for typing words of various lengths:

space, type, type, type

space, type

space, type, type, pause, type, type, type, type

Here, **space** represents the key event corresponding to pressing the space bar and **type** represents the key event with one key typed. These sequences represent typical usage where the user is typing from one word to the next.

Other common ways in which the user may start typing include: (i) using the cursor keys to navigate over to the desired position and then start typing, (ii) using a combination of meta keys (e.g., control, **Ctrl**) and the cursor keys to navigate over to the desired position and then start typing, and (iii) using the mouse to move over to desired area, then click into position, and start typing. The following sequences are examples of these three different ways of beginning a typing goal:

cursor, cursor, cursor, type, type, type, type, type

control+cursor, type, type, type, type, type

mouse, mouse, mouse, click, type, type, type, type, type

With automated help, the system can potentially save the user from having to type the letters for the rest of the words. This kind of help has the best potential for long words (reducing the required physical effort by the number of keystrokes needed otherwise) and

esoteric or domain-specific vocabulary (reducing the cognitive effort required to remember and spell the word accurately).

5.2 Finite State Automata

This section presents the deterministic finite state automata used in our goal recognition model. Section 5.2.1 presents the DFA we use in our goal model. With focus on the highlighting goal classes, we describe the event vocabulary in Section 5.2.2 and develop the corresponding templates and machines in Section 5.2.3. The stochastic component associated with each goal machine will be presented as part of the model in this section, but we do not discuss it in detail until we illustrate how it is used in the inference procedure in Section 5.3.

5.2.1 Model Set-Up

For illustration purposes, Figure 5.8 provides a partial template that shows the dynamics of recognizing highlighting goals with two font attributes without any automated help from the system. Starting from the state labeled `s0`, observing the select event `sel` enables the automaton to transition to the state `SELECTED`, while observing the event `sel(f1)` enables the automaton to transition to `SELECTED(F1)` where the selected string has `f1` applied to it already. Since this is a goal template, only generic attributes, such as `f1`, are used. (The use of generic attributes enables us to substitute different *instantiated* attributes (e.g., `bold`). We will see below that when attributes are instantiated in the goal machines, additional events will be added to the model.) From `SELECTED`, it is easy to see that the event `f1` enables the automaton to transition to `SELECTED(F1)`. Similarly, from `SELECTED(F1)`, the event `undo1` undoes the font attribute `f1` and enables the automaton to transition to `SELECTED`. In `SELECTED`, several events results in a self transition. For space reasons, we introduce several shorthand notations. Although we do not go into the

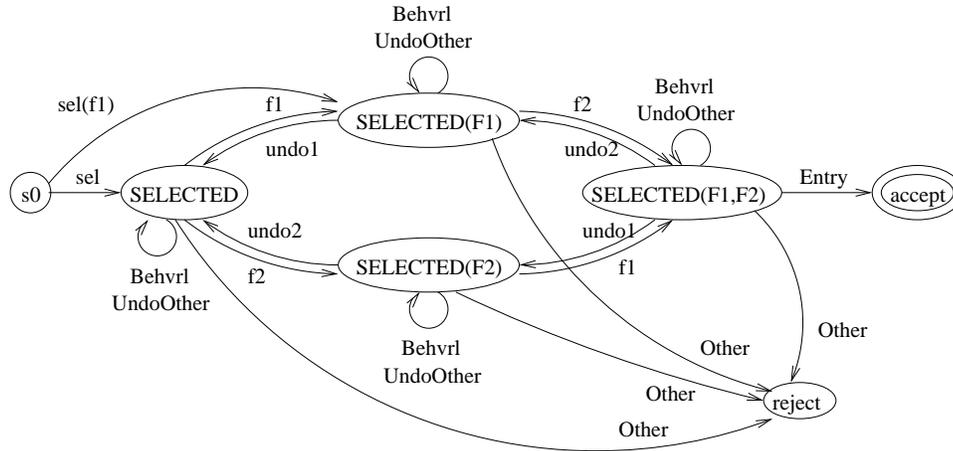


Figure 5.8: A partial highlighting template DFA using generic labels (e.g., `f1`) that can be used to instantiate a goal such as highlighting a string to be **bold** and **red**. Note that since templates use generic labels, this example shows the event `sel(f1)` but does not require `sel(f2)` to be explicitly represented. The explicit representation is only present in the instantiated goal machines.

details until Section 5.2.2, an event label beginning with a capitalized letter refers to a group of events. For example, the label `Behvrl` corresponds to three unique *behavioural* events. Other shorthand notation will be explained as they appear in the diagrams. The rest of this figure is interpreted similarly.

Next, we turn to the general model used to recognize goals. We adopt the DFA formalism to develop goal class templates and goal machines. A DFA is a tuple (Q, E, δ, Q_0, FS) where:

- Q is a finite, non-empty set of states
- E is the event vocabulary serving as the input symbols to the automaton
- $\delta : Q \times E \rightarrow Q$ is the transition function
- Q_0 is a start state s.t. $Q_0 \in Q$
- FS is a set of final states s.t. $FS \subseteq Q$

For the purposes of goal recognition modeling, we introduce additional notions to describe the state space. First, we introduce a special state that represents the user goal called the *goal state*. In essence, the goal state is the application state corresponding to the partial completion of the goal represented by the automaton. In Figure 5.8, the goal state is labeled as `SELECTED(F1,F2)`. Second, we define two final states — an *accept* state and a *reject* state — denoting when an event sequence is accepted or rejected respectively. These states have the same labels in Figure 5.8 also. Note that an accept state is different from a goal state because the system cannot be sure when a user goal is completed until an additional event is observed. The main reason is that the goal state of one automaton may be a non-final, non-goal state of another automaton. This situation is possible because goals are achieved via sequences of events, so different goals may be achieved by subsequences of each other. Therefore, the accept state is introduced as a way to designate the completion of a goal by observing an additional event. Third, we introduce terminology to capture the set of states relevant to cases that arise when an intelligent system offers help to the user. To model situations where the user may have accepted (partially) incorrect automated help, we define a set of *extraneous states*, ES , such that $ES \subset Q$. This name refers to the fact that these states involve events that are extraneous to the intended goal encoded in the goal state. Since Figure 5.8 is restricted to recognizing goals without automated help, the template shown does not include extraneous states.

An *event* is an abstracted user action. At the operating system level, user actions (when restricted to keyboard and mouse input devices only) are mouse clicks, mouse positions, and keystrokes. An event can be a user action as such, or a higher level event which is an aggregate of user actions, such as the selection of a string (abstracted from a mouse click, mouse drag movement, and a mouse button release). In Figure 5.8, we see examples such as `f1`, `undo1`, and behavioural events captured by `Behvr1`. The set of all events is referred to as the *event vocabulary*, E . Without going into the details

here, we enumerate these events for the relevant goal classes in the next section. Given a current (or *source*) state, an event, a transition is a tuple (q_t, e, q_{t+1}) , where $q_t, q_{t+1} \in Q$, $e \in E$, and t denotes the time step. In any given state in a DFA, all events transition to one and only one target state.

Both templates and machines adopt this definition of DFA, with the main difference being the elements of Q and E . For templates, the state descriptions are *generic* attributes used for parameterizing features of the goal class. Just as we saw in the simple template example in Figure 5.8, states and events involve labels such as **f1** rather than an actual name for the font attribute (such as **bold**). On the other hand, state descriptions for machines use *concrete* attributes to describe the actual feature executed as part of that goal. In essence, the generic attributes act like variables while the concrete attributes act as the instantiated values of the variables. For example, the template in Figure 5.8 may be instantiated for a highlighting goal with red and bold. In particular, all the state labels involving F1 is replaced by *bold*, and all the instances of F2 is replaced by *red*. Likewise, all the event labels involving **f1** is replaced by **bold**, and all the instances of **f2** is replaced by **red**.

In addition, goal machines extend the DFA definition with a stochastic component that models the probability of the user executing an event in each automaton state. In particular, the tuple defining goal machines is (Q, E, δ, Q_0, FS) , where we add:

- $Z : Q \times E \rightarrow \mathbb{R}$ is the observation model that represents the probability of an event in a given automaton state for that machine such that $\sum_{e \in E} Z(q, e) = 1$, $\forall q \in Q \setminus FS$.

All other components remain the same as above.

5.2.2 Event Vocabulary

Since events are input symbols to the DFA's designed for goal recognition, we model events that correspond to different kinds of user behaviour in the context of goal execution. Some behaviour will lead to initiating a goal activity, some will lead to progress in carrying out a goal or finishing a goal, while others may indicate behaviour specific to identifying user characteristics. In general, we developed four categories of user events to describe the different behaviour as they relate to the structure of the DFA. These categories are:

- **Entry Events:** These are goal-related events that allow transitions from the start state into an automaton. Intuitively, these events correspond to the first event that designates the start of a user goal at the beginning of an episode. The specific events we model depend on the goal classes of interest, which we enumerate fully below. As an example, selecting a string indicates the user is (potentially) starting a highlighting goal, thus, the select event is an entry event (although it is not an entry event for other goal classes such as initial indentation and multiple node-arrow goals).
- **Progress Events:** These are goal-related events that transitions toward or away from the goal state in an automaton. The specific events we model here also depend on the goal classes of interest. For example, the execution of font style events (e.g., red, bold, etc.) indicate progress for a highlighting goal, while undoing font style events may indicate regress from a highlighting goal. In cases where we need to distinguish between these two kinds of progress events, we will refer to them as *positive progress events* and *negative progress events*.
- **Response Events:** These events correspond to types of responses toward automated help, such as considering help, accepting help, and rejecting help. Note that the system may not be able to make suggestions that match the user's goal

perfectly, so accepting the automated help potentially results in transitioning to an automaton state different from the goal state. The specific events we model here also depend on the goal classes of interest. For example, from a pop-up toolbar, the user may select an icon whose font pattern matches that of the intended highlighting goal.

- **Behavioural Events:** In order to infer user characteristics, we define a set of events that capture behaviour indicative of the characteristics of interest (cf. Chapter 4). In this chapter, we restrict our attention specifically to the user's *level of neediness* and develop the associated events for recognizing it. Unlike the other event categories, behavioural events modeled are independent of goal classes, although they could depend on the application or input devices used. Examples of behavioural events include a user using the mouse to hover over the application interface to look for help, or pausing because he is unsure of what to do.

In defining our event vocabulary, we focus on the highlighting and typing goal classes as described in Section 5.1.1 and Section 5.1.7 respectively. Our objective is to use highlighting goals to demonstrate the goal recognition component, with the typing goals serving to provide context during the interaction. In particular, we consider highlighting goals with *at most* four font attributes, both for users executing highlighting goals as well as for the system suggesting highlighting help. We present the events first for highlighting templates, then for highlighting machines, and lastly for typing goals.

5.2.2.1 Events for Highlighting Templates

As presented in the example event sequences from Section 5.1.1, the relevant events are selecting a string, applying font attributes, undoing font attributes, considering help, accepting help of a certain combination of font attributes, and various behavioural events. Table 5.2 lists the generic events used for highlighting templates. This table is separated in four columns corresponding to each event category. The top section of the table

Table 5.2: Events for the highlighting goal class templates.

Entry	Progress	Response	Behavioural
<code>sel</code>	<code>f1, f2, f3, f4,</code>	<code>cons</code>	<code>pause</code>
<code>sel(f1)</code>	<code>f5, f6, f7, f8</code>	<code>acc(f1)</code>	<code>browse</code>
<code>sel(f1, f2)</code>	<code>undo1, undo2, undo3, undo4,</code>	<code>acc(f1, f2)</code>	<code>waver</code>
<code>sel(f1, f2, f3)</code>	<code>undo5, undo6, undo7, undo8</code>	<code>acc(f1, f2, f3)</code> <code>acc(f1, f2, f3, f4)</code>	
<code>sel(f1, A)</code>	<code>undoA, undoB, undoC</code>	<code>acc(f1, A)</code>	
<code>sel(f1, A, B)</code>		<code>acc(f1, A, B)</code>	
<code>sel(f1, A, B, C)</code>		<code>acc(f1, A, B, C)</code>	
<code>sel(f1, f2, A)</code>		<code>acc(f1, f2, A)</code>	
<code>sel(f1, f2, A, B)</code>		<code>acc(f1, f2, A, B)</code>	
<code>sel(f1, f2, f3, A)</code>		<code>acc(f1, f2, f3, A)</code>	

corresponds to events for typical execution of highlighting goals, while the bottom section corresponds to events associated with extraneous states designed to handle scenarios of users accepting incorrect help.

When a string is selected, we model the cases when the user selects a string with plain, default font attributes using `sel`, as well as the cases when the user selects a string that already has one, two, or three font attributes applied to it using `sel(f1)`, `sel(f1, f2)`, and `sel(f1, f2, f3)` respectively. Symbols such as `f1` and `f2` are generic events that refer to two different system font events, such as red and bold. (The specific instantiation is given below when we discuss the events for highlighting machines.) The first select event, `sel`, is used to model when the user wants to start a highlighting goal, while the last three select events allow us to model the scenarios where the user wants to

change the highlighting pattern of a previously completed phrase. In addition, we model situations where users select a string that already has font attributes applied to them, but intend to modify some or all of those attributes using select events in the bottom part of the table: `sel(f1,A)`, `sel(f1,A,B)`, etc. The attributes A, B, and C are used to denote extraneous attributes that are not part of the goal. We will see specific examples below to show how these are used.

Overall, the system models eight font attribute events `f1`, `f2`, ..., `f8`, and eight corresponding events that undo each font attribute `undo1`, `undo2`, ..., `undo8` respectively. Additional font attributes can be modeled by increasing the number of font application and undo events.

We consider automated help to be provided via a suggestion toolbar with a set of macro icons, where each icon serves as a shortcut to executing a distinct highlighting goal. When the toolbar appears, the user may pause to consider the automated help offered. This scenario is modeled by the event `cons`. Accepting help is equivalent to selecting an icon from the suggested toolbar. When the user selects an icon, the font attributes applied to the selected phrase will be the attributes encoded in the icon. Restricting our attention to icons with a maximum of four font attributes in each combination, we have a set of response accept events `acc(f1)`, ..., `acc(f1,f2,f3,f4)`.

In addition, we model behavioural events that indicate the user's neediness level — `pause`, where the user is pausing to look for help or to figure out what to do next, `browse`, where the user is actively browsing for help but not selecting any specific function from the application interface, and `waver`, where the user is changing his mind and has executed an above average number of delete or undo keystrokes in recent event sequences.

As mentioned earlier, the bottom part of Table 5.2 lists the events associated with extraneous states. For example, while the intended goal may be red and bold, the user incorrectly selected suggested help with red, bold, and italics from the automated system. Consequently, the user may undo the undesired, or extraneous, attributes — italics, in

Table 5.3: Example events for the highlighting goal machines.

Entry	Progress	Response	Behavioural
<code>sel</code>	<code>red</code>	<code>cons</code>	<code>browse</code>
<code>sel(red)</code>	<code>italics</code>	<code>acc(red)</code>	<code>pause</code>
<code>sel(bold)</code>	<code>bold</code>	<code>acc(pink)</code>	<code>waver</code>
<code>sel(bold,underline)</code>	<code>undo-red</code>	<code>acc(bold,italics)</code>	
<code>sel(red,bold,underline)</code>	<code>undo-bold</code>	<code>acc(red,bold,italics)</code>	
<code>:</code>	<code>:</code>	<code>:</code>	

this case — and apply the font attributes that lead to the correct goal pattern. Extraneous attributes are defined relative to a goal pattern. Since we restrict our attention to highlighting goals with a maximum of four font attributes, we model up to three extraneous attributes per accepted help. We label extraneous attributes as one of A, B, or C. To model the interaction described in the aforementioned example, we add accept events that allow users to accept icons with partially incorrect goal patterns and undo events that allow users to undo extraneous attributes. Specifically, we include the following undo events `undoA`, `undoB`, and `undoC`, and accept events `acc(f1,A)`, ..., `acc(f1,f2,f3,A)`, which are enumerated in Table 5.2.

5.2.2.2 Events for Highlighting Machines

Highlighting machines are instantiated from a highlighting template. Although generic font attributes are used in event labels for highlighting templates, the system keeps track of all the specific, observed events in each episode and uses these events to instantiate user-specific goal machines. Thereafter, these machines are used to infer the current user goals. A partial list of example machine events is shown in Table 5.3.

As evident in the list of example machine events in Table 5.3, the only difference

Table 5.4: Events for the typing goal class.

Entry	Examples
space	space key
click	mouse click
cursor	left cursor keypress, control-cursor combination keypresses
Progress	Examples
type	letter keypress
delete	backspace or delete keypress over a letter

between the event set for templates and machines is that template events with generic or extraneous attributes become instantiated with concrete attributes. For example, the template events `undoA` and `f1` may become `undo-red` and `bold` respectively. Overall, the eight font attributes we consider are: `red`, `bold`, `italics`, `underline`, `shadow`, `burgundy`, `pink`, and `blue`. As an example, the generic event `sel(f1,f2)` may be instantiated as `sel(red,bold)`, or `sel(red,italics)`, or `sel(underline,italics)`, etc. The possible combinations depend on the specific machines stored in the library. Likewise for accept events, the combination of generic and extraneous attributes also result in many more combinations of machine events.

5.2.2.3 Events for Typing Goals

Since our interest is to model slide authoring interactions, we define events to recognize typing goals as discussed in Section 5.1.7. Because highlighting goals are the main focus of the goal model, we do not develop machines for recognizing typing goals. In other words, the present system will not offer help for typing and no response events are needed. The goal-related events for typing are listed in Table 5.4.

5.2.3 Highlighting Templates and Machines

Recall from Section 5.1.1 that highlighting goals changes the font attributes of a selected target string to match those of a target goal pattern. In particular, a target pattern is parameterized by the number of font attribute changes, relative to the default font settings in the slide presentation. Let K represent this parameter in the target pattern. For example, if the target pattern is red and bold, then $K = 2$, and if the target pattern is red, bold, italics, then $K = 3$. As mentioned in Section 5.2.1, we define automata states as application states. Since the automata models the user's progress toward the goal state (i.e., the state representing the target pattern), the number of states in the automata grows as a function of K . Below, we present the highlighting templates for the cases of $K = 1$ and $K = 2$ in detail and describe the templates for more complex cases (i.e., $K > 2$). Thereafter, the corresponding goal machines will be described. To reiterate, we assume that the maximum value of K is 4 for highlighting goals that are executed by the user or from the system's icon suggestions.

5.2.3.1 Templates for the Case of $K = 1$

We first illustrate the highlighting template with $K = 1$ font attribute changes, starting with the scenario when no automated help is offered. This partial template is shown in Figure 5.9. The template has five states: `s0`, `SELECTED`, `SELECTED(F1)`, `accept`, `reject`. The first state is the start state, the next two states represent the progress toward the user goal, reflecting that zero or one font attribute has been applied to it, with the latter being the goal state. The last two states are final states. The state `SELECTED` indicates that a string with default font pattern has been selected, and the goal state `SELECTED(F1)` indicates that the selected string has a single font attribute, generically referred to as `F1`, applied to it. Transitioning from `SELECTED` to `SELECTED(F1)` reflects the shortest path from to the goal state. The template also models less efficient dynamics using self-loops when behavioural events and negative progress events (such as undo events) are observed.

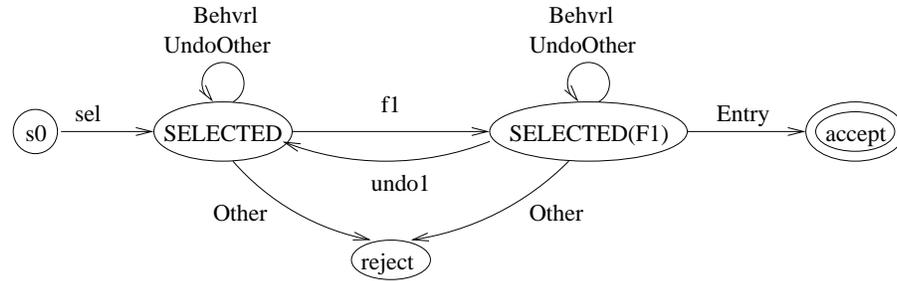


Figure 5.9: Partial highlighting template for $K = 1$ when no automated help is present.

For notational convenience, we introduce the following abstract events: **Behvrl** refers to any behavioural event defined in the vocabulary, **Entry** refers to any entry event defined in the vocabulary (regardless of the goal class), **UndoOther** refers to other undo events not already mentioned from a given state in that automaton, and **Other** refers to all other events not already mentioned as an outgoing transition from a given state in that automaton.

Next, we extend the partial template in Figure 5.9 to model the dynamics with automated help shown in Figure 5.10. Four changes are made to the partial template. First, we include the response event **cons** that denotes the user considering help in all the states as a loop transition because this event is, in effect, a **pause** but only in the context when system help is available. Second, we add the accept event **acc(f1)** that transitions to the goal state. In general, we assume automated help is available at every time step so that the user may accept suggestions from any non-final state after entering the machine. For example, if the current state is **SELECTED** and the user accepts help from the system, the machine would transition directly to the goal state via an **accept** event. The same applies to other non-final states, except for **s0** because no string is selected yet (recall that the system only makes highlighting suggestions after a phrase has been selected). Thus, to extend the automaton, we add a new transition from every non-final state (excluding **s0**) to the goal state via the event **acc(f1)**. To avoid clutter in the diagrams, we introduce the shorthand notation “*NF” to denote any non-final state

attributes in order to backtrack to the goal state. Their labels are interpreted analogously to the other states; `SELECTED(F1,A)` indicates that the selected string has the desired font attribute and one extraneous attribute applied to it, `SELECTED(F1,A,B)` indicates that the selected string has the desired font attribute and two extraneous attributes applied to it, etc.

The last modification is to add select events that involve extraneous attributes. In particular, for the template case of $K = 1$, we add `sel(f1,A)`, `sel(f1,A,B)`, and `sel(f1,A,B,C)`. These events are analogous to the events that accept into the extraneous states. Thus, they follow the same transition dynamics of undoing the undesired attributes to get to the goal state. Figure 5.10 illustrates the completed highlighting template with these modifications for the case of $K = 1$.

The extraneous states in this template model a subset of the recovery scenarios after the user accepts partially correct help from the system. Although there are other, more complicated, sequences of events that the user may execute to indicate that he is still interested in achieving a highlighting goal after accepting partially correct help, we consider them unlikely and do not model them here.

5.2.3.2 Templates for the Case of $K = 2$

Next, we illustrate the highlighting template with $K = 2$ beginning with the partial template without automated help in Figure 5.11. This template has seven states: `s0`, `SELECTED`, `SELECTED(F1)`, `SELECTED(F2)`, `SELECTED(F1,F2)`, `accept`, `reject`. The first is a start state, the next four states are non-final states, with the fourth one being the goal state, state, while the last two are final states. The dynamics of this template is interpreted analogously to that of the template for $K = 1$, with `SELECTED(F1,F2)` indicating a selected string has two font attributes, `F1` and `F2`, applied to it. These non-final states show two shortest paths to get from `SELECTED` to `SELECTED(F1,F2)` — either by applying `f1` first and then `f2` or the other way around. An extra entry transition,

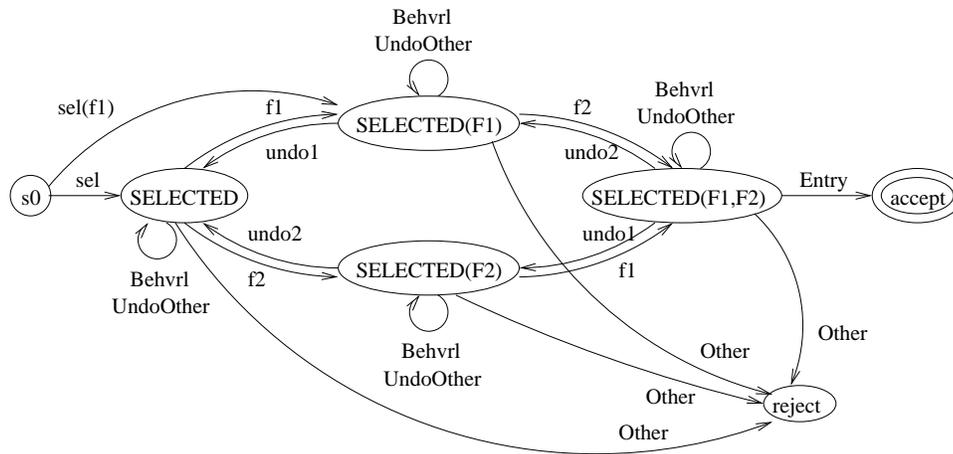


Figure 5.11: Partial highlighting template for $K = 2$ when no automated help is present.

$\text{sel}(f1)$ from s_0 , is modeled to allow users to select a previously highlighted string and change its target goal pattern to involve an additional font attribute. Additional transitions that allow less efficient transitions to the goal state, such as the self-loops and undos, are also modeled in this template.

To model automated help response behaviour, we extend the partial template in Figure 5.11 to the full template in Figure 5.12. Recall from the case of $K = 1$ that four changes were added to the partial template: (i) add cons to each non-final state as a self-loop to model the user considering help, (ii) add appropriate accept events for the basic template to indicate the user accepting certain kind of help, (iii) add extraneous states and associated accept events to model behaviour for accepting partially correct help and changing the pattern to get back to the goal state, and (iv) add select events involving extraneous attributes to model behaviour for modifying previously completed goals. The first modification is carried out in the same way as before. The second involves adding the events $\text{acc}(f1)$, $\text{acc}(f2)$, and $\text{acc}(f1,f2)$ from any non-final state to transition to the appropriate non-final state. The third modification involves creating extraneous states, accept events, and undo events based on having accepted partially correct help of up to two extraneous attributes via the event $\text{acc}(f1,f2,A,B)$. The details of this step

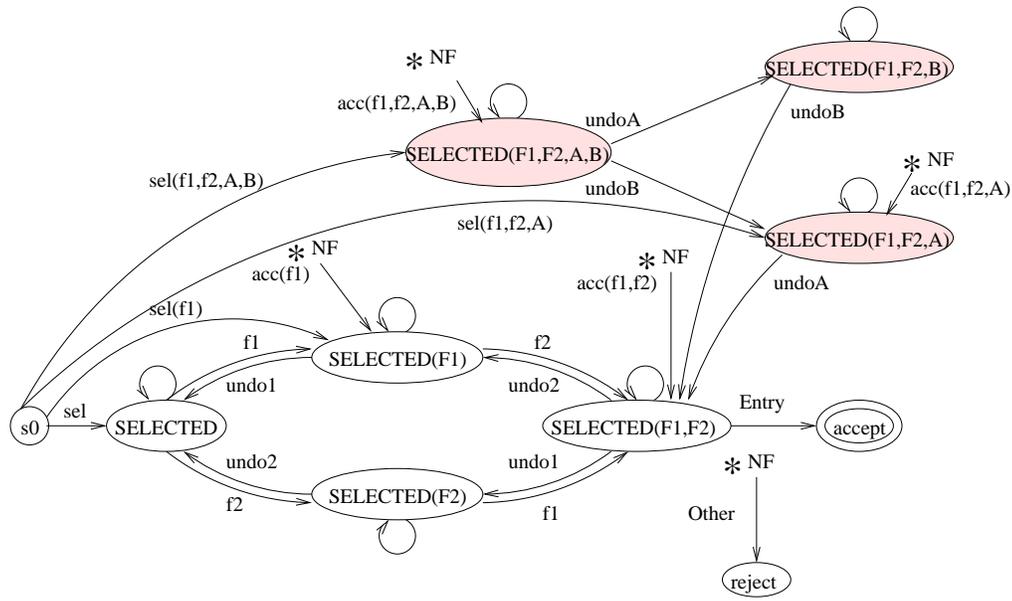


Figure 5.12: Highlighting template for $K = 2$, extended from the basic template drawn in Figure 5.11. For space reasons, loop transitions are not labeled, although they are the same as those in Figure 5.11.

are carried out in the same way as before. Select events analogous to the accept events included in the third modification are added as the last change. The extended template is illustrated in Figure 5.12.

5.2.3.3 Templates for the Case of $K > 2$

The creation of highlighting templates for $K > 2$ follow the same steps as described above. Thus, we present the templates for $K = 3$ and $K = 4$ without going into details. Figure 5.13 shows the partial template for $K = 3$ without automated help, with the goal state as $\text{SELECTED}(F1,F2,F3)$.

In Figure 5.13, we have events for selecting a string, applying font attributes, undoing them, and executing needy behaviour. Analogous to the case of $K = 2$, we model entry events sel , $\text{sel}(f1)$, and $\text{sel}(f1,f2)$, so that previously completed goals may be modified. Since templates refer to generic attributes, we do not need to model the event

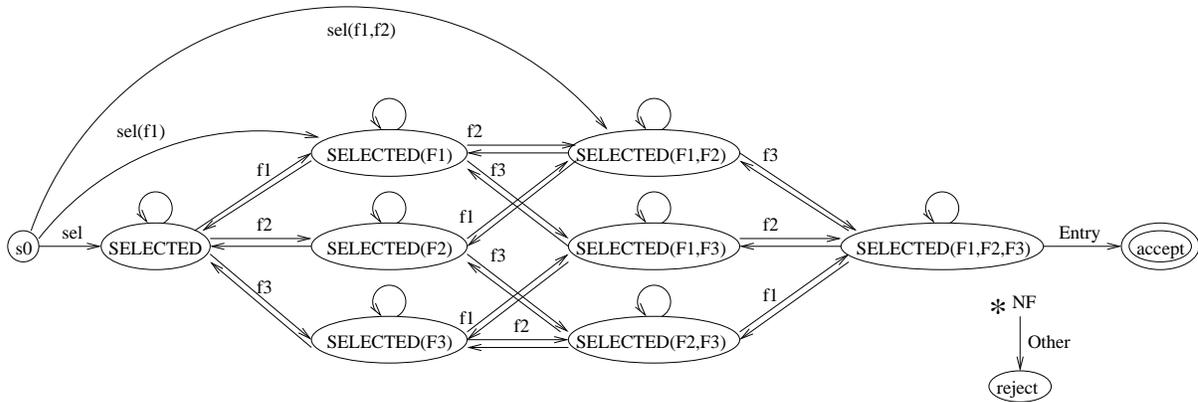


Figure 5.13: Partial highlighting goal class template $K = 3$ when no automated help is present. For space reasons, loop transitions and negative progress events are not labeled.

$sel(f2)$ because a string with a single attribute can be relabeled as $sel(f1)$ by the system. The rest of the template is interpreted in the same manner as the previous ones.

To extend this partial template to include automated help, we add $cons$ as a self-loop in all the non-final states and $acc(f1)$, $acc(f1,f2)$, and $acc(f1,f2,f3)$ transitioning from all the non-final states to $SELECTED(F1)$, $SELECTED(F1,F2)$, and $SELECTED(F1,F2,F3)$ respectively. In considering users accepting partially correct help, we include the event $acc(f1,f2,f3,A)$ and the extraneous state $SELECTED(F1,F2,F3,A)$. Finally, the event $sel(f1,f2,f3,A)$ is also added. The final template for $K = 3$ is illustrated in Figure 5.14.

The last case we present is the highlighting template for $K = 4$. Because of our restriction that highlighting goals and suggestion macros are limited to a maximum of 4 font attributes, this template does not have extraneous states or events with extraneous attributes. For this reason, the template without automated help has the same structure as the final template for $K = 4$, with the exception that the final template models response events. In Figure 5.15, we illustrate the final template for $K = 4$, where the goal state is $SELECTED(F1,F2,F3,F4)$. As shown in previous cases, our highlighting templates, parameterized by K attributes in the target goal pattern, has K entry events that lead to

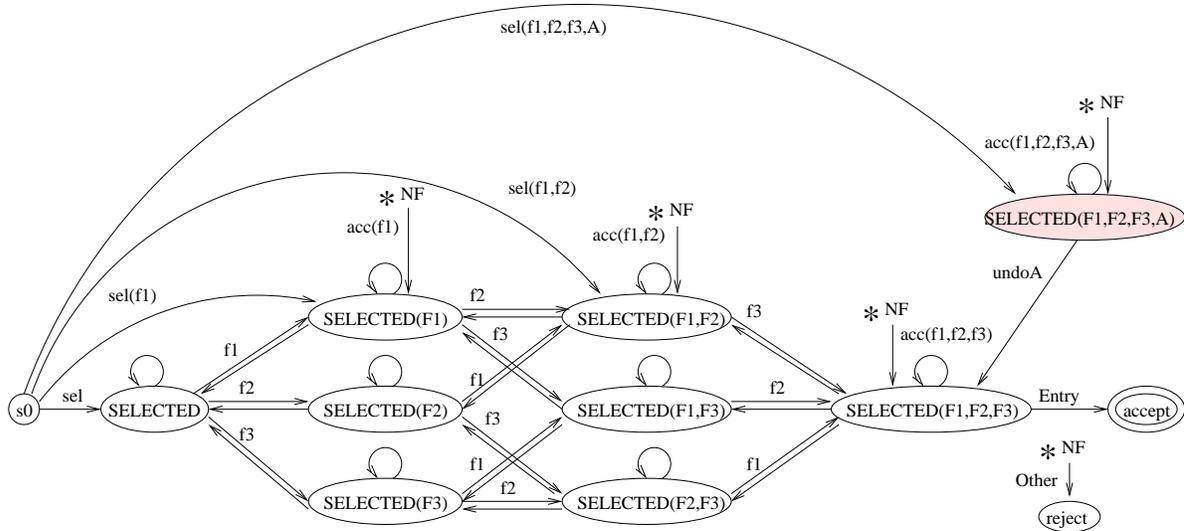


Figure 5.14: Highlighting template for $K = 3$, extended from the partial template drawn in Figure 5.13. For space reasons, loop transitions and negative progress events in the basic template are not labeled.

a non-final state. In this case, we have sel , $sel(f1)$, $sel(f1, f2)$, and $sel(f1, f2, f3)$. This automaton also models $cons$ as a self-loop in all the non-final states, and $acc(f1)$, $acc(f1, f2)$, $acc(f1, f2, f3)$, and $acc(f1, f2, f3, f4)$ that transition from any non-final state to the state with the corresponding combination of font attributes accepted. The rest of the template is constructed and interrupted in the same way as in the previous cases.

The four templates for $K = 1, \dots, 4$ presented above make up the inventory of goal class templates in our system.

5.2.3.4 Template Instantiation: Highlighting Machines

Recall the goal recognition architecture from Figure 5.1. Each user event is processed as an incoming event by the machines in the library. The transition dynamics of these machines follow that of DFA's. A highlighting goal is recognized as completed when either one of the machines in the library arrives at an accept state, or if the episode's

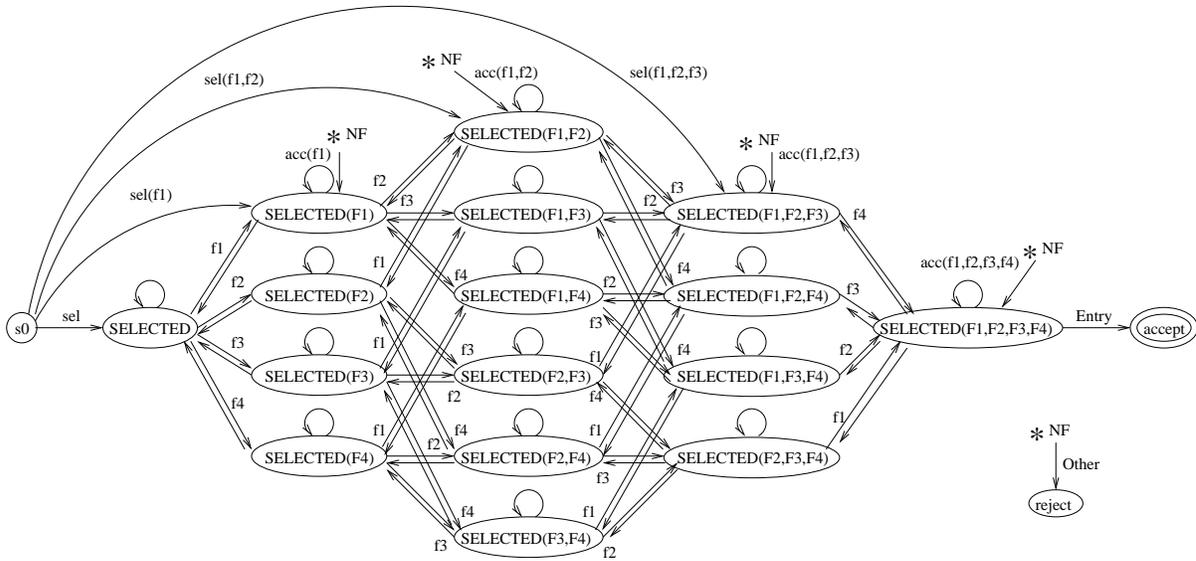


Figure 5.15: Highlighting template for $K = 4$. For space reasons, loop transitions and negative progress events are not labeled.

interaction history indicates a new highlighting goal has been completed. When a goal is completed, the system takes the interaction history, determines the highlighting goal that was executed (if any), determines whether a highlighting machine already exists in the library for the completed goal, and if needed, instantiates a new goal machine using the appropriate template and adds the machine to the library. When a new machine is instantiated, the system updates the goal distribution based on the observed interaction history, and the recognition process repeats. For illustration purposes, we present some examples of the instantiated machines.

The first example we consider is the case of a highlighting goal with four font attributes, resulting from observing the following sequence of user events:

`select, red, bold, italics, shadow`

The resulting machine that is instantiated is shown in Figure 5.16.

In comparison to the template in Figure 5.15, the main differences in this machine are:

- (i) references to generic font attributes in the state labels are replaced by the observed

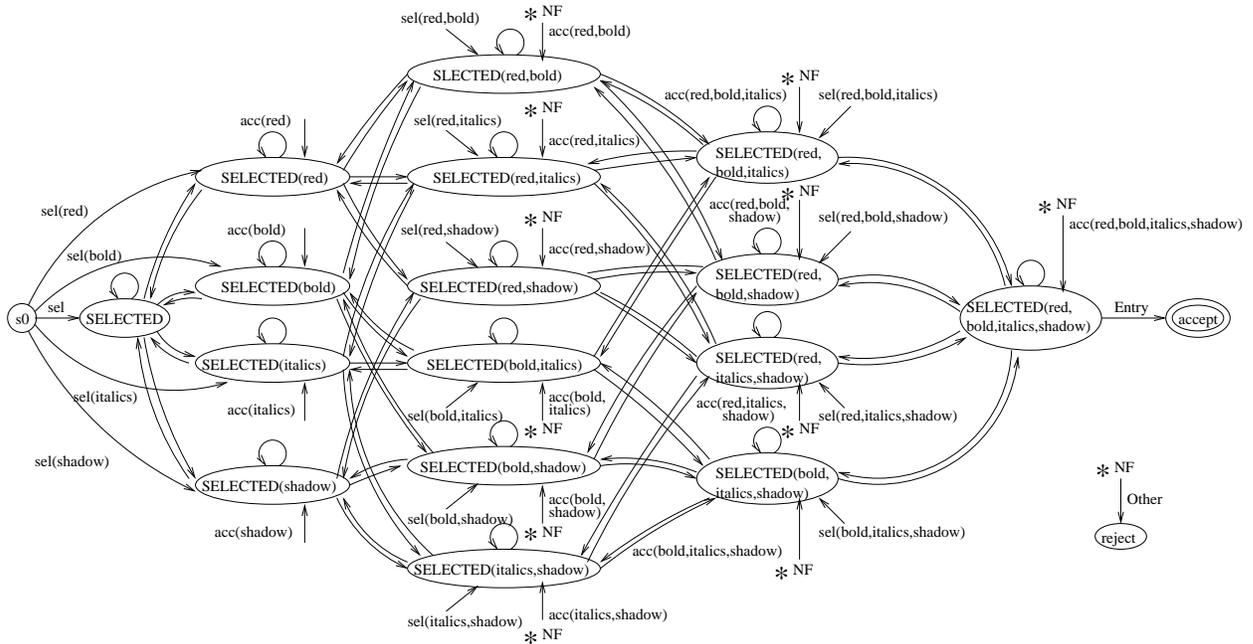


Figure 5.16: Highlighting machine for the goal `red,bold,italics,shadow`. For space reasons, progress events and loop transitions are not labeled. To minimize clutter, entry transitions do not all connect to `s0`.

ones, (ii) multiple copies of select and accept events are created, and (iii) references to generic font attributes in the event labels are replaced by the observed ones. Examples of changing the state labels is that `SELECTED(F1)` becomes `SELECTED(RED)`, `SELECTED(F1,F2)` becomes `SELECTED(RED,BOLD)`, etc. An example of making multiple select events and having their generic labels replaced is that `sel(f1)` becomes `sel(red)`, `sel(bold)`, `sel(italics)`, and `sel(shadow)` to indicate that any one of the attributes of the target pattern could be selected.

As mentioned above, our automata distinguishes between a goal state and an accept state because the system cannot be sure that a goal is completed until the next event is observed. In particular, if the goal has ended, then that next event must be an entry event because a new goal has begun. In Figure 5.16, the transition after the goal state represents either a typing goal or another highlighting goal has begun. Therefore, when

Table 5.5: Interaction history for the goal `red,bold,italics,shadow`.

Previous State	Event	Next State
s0	<code>select</code>	SELECTED
SELECTED	<code>red</code>	SELECTED(REDD)
SELECTED(REDD)	<code>bold</code>	SELECTED(REDD,BOLDD)
SELECTED(REDD,BOLDD)	<code>italics</code>	SELECTED(REDD,BOLDD,ITALICSD)
SELECTED(REDD,BOLDD,ITALICSD)	<code>shadow</code>	SELECTED(REDD,BOLDD,ITALICSD,SHADOWD)

one of the machines in the library arrives at its accept state, it means that machine's goal has been completed and an additional entry event has been observed. At this point, the system updates its library and goal distributions with the episode's interaction history. Since a new goal has begun, all the machines are *reset* back to its start state, s0, and that additional entry event is re-used as the first event of the next episode. In other words, the last event of every episode is *recycled* as part of the interaction of the next episode.

The above sequence of user events gives us a partial history of the interaction that occurs in executing the highlighting goal. Reviewing the architecture of the goal recognition component from Figure 5.1, this step takes place in the procedure that stores incoming user events from the current episode. The complete history our system models is a list of state-event-state tuples corresponding to the transitions represented by the event sequence and the resulting machine. In particular, the transitions corresponding to the event sequence above are listed in Table 5.5. Note that this history does not include any transitions executed beyond the goal state.

This history represents a particular way in which the user executed this goal. In the long run, if the user has general tendencies to execute events in certain order, then we would like to use this information to help make better predictions about the user's goal. We model this behaviour because we believe that these patterns may reflect the

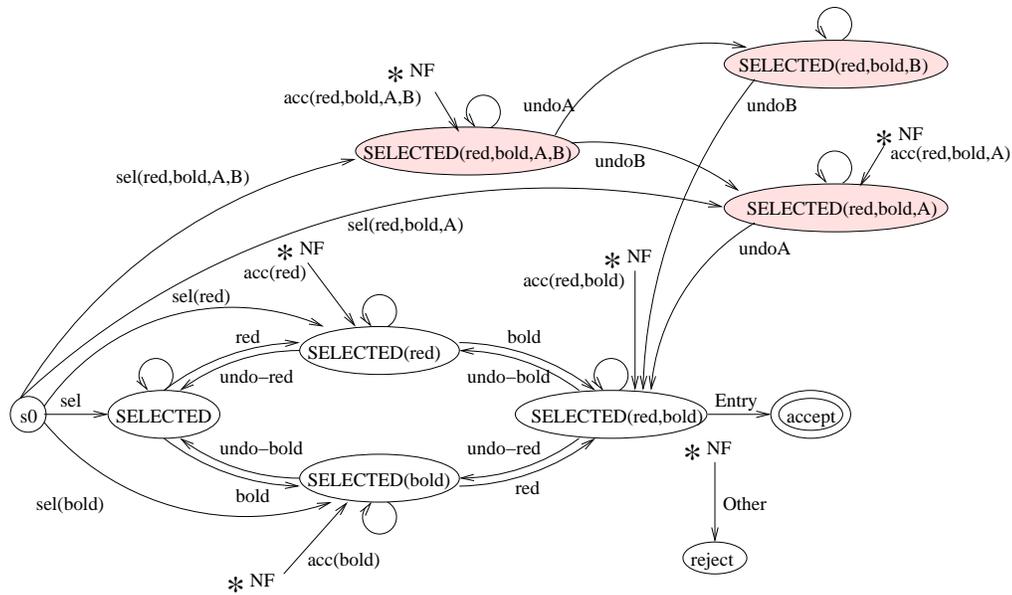


Figure 5.17: Highlighting machine for the goal `red, bold`. Labeling and notation follow that of Figure 5.16.

user’s preferred way of executing those goals. Therefore, our system uses the interaction history to update the goal distribution. The details of this update process are discussed in Section 5.3. Here, we focus our discussion on how the interaction history is created given the observed event sequence and the resulting, instantiated machine.

Next, we consider a less trivial example in the case of a highlighting goal with two font attributes, resulting from the following sequence of events:

`select, pause, browse, red, cons, bold`

These events indicate that the user selected a string, paused and browsed around the application interface (without selecting any widget), applied the red attribute to the selected string, considered suggested help, and then applied the bold attribute to the selected string. The resulting machine that is instantiated is shown in Figure 5.17.

The machine in Figure 5.17 is instantiated by following the same steps of creating copies of select and accept events and replacing generic font attribute labels with observed

Table 5.6: Interaction history for the goal `red,bold`.

Previous State	Event	Next State	Keep?
s0	<code>select</code>	SELECTED	No
SELECTED	<code>pause</code>	SELECTED	No
SELECTED	<code>browse</code>	SELECTED	No
SELECTED	<code>red</code>	SELECTED(RED)	Yes
SELECTED(RED)	<code>cons</code>	SELECTED(RED)	No
SELECTED(RED)	<code>bold</code>	SELECTED(RED,BOLD)	Yes

ones. Note that the extraneous attribute labels in this machine (for both state and event labels) are left uninstantiated because they are designed to represent any potential extraneous attributes that may be changed or inadvertently accepted by the user in future episodes. In other words, labels for extraneous events and states are instantiated and updated dynamically during the recognition process.

The sequence of user events observed in this example involves behavioural and response events. Table 5.6 shows the interaction history corresponding to this event sequence and the instantiated machine. Because we are interested in modeling how the user executes highlighting goals, our update process for the goal distribution only models the partial history that is associated to goal execution. For example, if the user paused or accepted help, those events do not contribute to how the user executed the goal. Thus, Table 5.6 displays an extra column that indicates whether the system will keep the state-event-state tuple shown in that row in updating the goal distribution.

With respect to the goal recognition architecture presented in Figure 5.1, the step for discarding a subset of the history takes places in the procedure that parses the episode history to obtain goal events. The interaction history shown in Table 5.6 indicates that a machine with `red,bold` is created (if it does not already exist in the library), and that

Table 5.7: Interaction history for the first event sequence example.

Previous State	Event	Next State	Keep?
s0	<code>select(bold)</code>	SELECTED(BOLD)	No
SELECTED(BOLD)	<code>red</code>	SELECTED(RED,BOLD)	Yes

Table 5.8: Interaction history for the second event sequence example.

Previous State	Event	Next State	Keep?
s0	<code>select</code>	SELECTED	No
SELECTED	<code>acc(red,bold)</code>	SELECTED(RED,BOLD)	No

the behaviour of executing `red` before `bold` is modeled in the goal distribution. Using the target pattern `red,bold` that results in the instantiated machine from Figure 5.17, we consider additional scenarios where the system discards a subset of the episode history. The first example sequence is:

```
select(bold), red
```

Here, the string `selected` has been applied `bold` previously, and the user modifies it by adding `red`. The corresponding interaction history is shown in Table 5.7. As indicated, the first tuple involving a previously executed goal is not included as part of the execution pattern of this goal.

The second example consists of the following event sequence whose interaction history is shown in Table 5.8:

```
select, acc(red,bold)
```

We do not consider accepting automated help as part of the goal execution behaviour of the user because the font attributes accepted together in a single suggestion do not

provide any information about the user's preferences over the order of event execution. Thus, this example illustrates that we exclude the last tuple of the interaction history when we are updating our goal distributions.

A related example involves an event sequence with multiple accept events in the episode:

```
select(shadow), acc(blue), undo-blue, italics, bold, acc(red,bold)
```

The last instance of the accept event indicates that the desired goal is `red,bold`. We interpret this behaviour to mean that all the behaviour prior to this event is irrelevant to the ultimate goal desired by the user. To handle this scenario, we eliminate the events prior to the last instance of accept, and to replace it with a simple `select` event only. The resulting event sequence is:

```
select, acc(red,bold)
```

The corresponding interaction history and the partial history that is kept for updating the goal distribution is the same as that shown in Table 5.8.

The last example we consider involves the user undoing font attributes illustrated in the following event sequence:

```
select, shadow, bold, undo-shadow, red
```

Since the desired goal is `red,bold`, the history we consider excludes the `shadow` and `undo-shadow` events. Thus, the event sequence we obtain is:

```
select, bold, red
```

This sequence is then used with the instantiated machine to create the interaction history shown in Table 5.9.

In summary, the history we use to update the goal distributions pertain to the execution of the current font attributes only; this excludes the selection of the string to be

Table 5.9: Interaction history for the third event sequence example.

Previous State	Event	State	Keep?
s0	select	SELECTED	No
SELECTED	bold	SELECTED(BOLD)	Yes
SELECTED(BOLD)	red	SELECTED(RED,BOLD)	Yes

highlighted, the consideration or acceptance of automated help, behavioural events, behaviour prior to the last instance of accepting automated help, and behaviour associated with undoing a part of a goal.

5.3 Inferring the User’s Current Goal

In this section, we describe the third part of the goal recognition component from Figure 5.1 that infers the user’s current goal. In Section 5.3.1, we define the task of interest, with reference to the goal distributions (the episode prior, the event prior, and the event posterior). As part of the calculations involved in this inference task, we use empirical frequencies to update two components: the observation model associated with each goal machine and the episode prior goal distribution. The steps involved in updating these are explained in Section 5.3.2 and Section 5.3.3 respectively.

5.3.1 The Inference Task

Let G be the library of goals and g_1, \dots, g_N be the set of N goals in the library at a given point in time. To infer the user’s goal, we compute the probability of a goal in the library, given the episode’s interaction history observed so far, $Pr(G|EV_1, \dots, EV_t)$. According to the model of our goal machines, each machine can only be in one state at any given time. Recall the DFA notation that Q defines the set of states in an automaton. Following this

notation, let us define Q^{g_i} to denote the set of states for goal g_i , for any $g_i \in G$. Since our goal model is a deterministic automaton, only one state in each goal machine has non-zero probability at any given point in time. In other words, for all the states in a goal machine $q_1^{g_i}, q_2^{g_i}, \dots$ where each $q_l^{g_i} \in Q^{g_i}$ for any l , only one such state has non-zero probability. Let us refer to this state with non-zero probability as $q_{l^*}^{g_i}$. Then, the probability of g_i being the intended user goal is simply the probability of being in $q_{l^*}^{g_i}$. To express this, let us define MS to be the set of machine states for all the machines in the library; that is, $MS = Q^{g_1} \cup \dots \cup Q^{g_N}$ for a library with N goals. Note that since machine states are unique to each goal, the size of MS is the sum of the number of states in all the goal machines. Then, the probability that the current machine state is really $q_{l^*}^{g_i}$ is simply the probability that the current user goal is g_i , i.e., $Pr(MS = q_{l^*}^{g_i}) = Pr(G = g_i)$. Based on this relationship, we recast our inference task of interest as $Pr(MS_{t+1}|EV_1, \dots, EV_t)$.

We compute $Pr(MS_{t+1}|EV_1, \dots, EV_t)$ in an incremental fashion. At time $t = 0$, all the machines in the library are beginning a new episode. Thus, each machine is set to its start state, s_0 . This distribution defines $Pr(MS_0)$ when no observations are present. At time $t > 0$, we use the transition and observation models to update the distribution as defined in Equation (5.1):

$$Pr(MS_{t+1}|EV_t) \propto \sum_{MS_t} Pr(EV_t|MS_t)Pr(MS_{t+1}|MS_t, EV_t)Pr(MS_t) \quad (5.1)$$

where $Pr(EV_t|MS_t)$ is the observation model (described below), $Pr(MS_{t+1}|MS_t, EV_t)$ is the deterministic transition model as defined by the goal machines in the system library, and $Pr(MS_t)$ is the distribution over machine states at the previous time step. The quantity $Pr(MS_t)$ is available from the previous time step. The observation model is learned empirically to reflect individual usage as we describe below in Section 5.3.2. The transition model is defined according to the transition dynamics in the goal machines described in Section 5.2.3, with $Pr(MS_{t+1}|MS_t, EV_t)$ simply being the aggregate (“stacked”) model of $Pr(Q_{t+1}^{g_1}|Q_t^{g_1}, EV_t)$, ..., and $Pr(Q_{t+1}^{g_N}|Q_t^{g_N}, EV_t)$. In particular, since MS is the union of all the goal machine states, when $ms_t = q_{j,t}^{g_i}$, referring

to the j^{th} automaton state in the goal machine for g_i at the t^{th} time step, we define $Pr(MS_{t+1}|ms_t, EV_t) = Pr(Q_{j,t+1}^{g_i}|q_{j,t}^{g_i}, EV_t)$, for all events EV_t and for all states of $Q_{j,t+1}^{g_i}$. When MS_{t+1} refers to a state not in the goal machine for g_i then the probability is zero (i.e., when transitioning from state g_i to one out of g_i).

5.3.2 Empirically Updating the Observation Model

As mentioned, the observation model $Pr(EV_t|MS_t)$ is empirically updated using the episode's interaction history. Each goal machine has an associated observation model that records the empirical probability of an event given any automaton state, for all events defined in the vocabulary. These empirical probabilities are estimated based on the observed interaction history in all previous episodes. Recall from Section 5.2.3 that certain portions of the interaction history are kept or discarded in updating the goal distribution. The partial history that is kept is used to update the empirical counts in the observation model in this step as well. Thereafter, we apply a simple add-one smoothing to avoid have automaton state-event pairs with zero probability and normalization to obtain the actual probability value. More specifically, let the un-smoothed probability be $pr(ev|ms) = \frac{c(ev,ms)}{N}$, where C corresponds to the counts of observing ev in a given ms , and N is the total number of times that any event has been observed in ms . Then, the smoothed probability is $pr_{+1}(ev|ms) = \frac{c(ev,ms)+1}{N+V}$, where V is the number of events.

5.3.3 Empirically Updating the Episode Prior

The goal distribution we focus on in this section is the episode prior, $Pr(G)$. Since the user may be interested in carrying out a goal other than one that is known to the system, we model an abstract goal, g_0 , to represent any other goal not present in the system's library. Thus, with N goals stored in the library, $Pr(G)$ is a distribution over $N + 1$ goals. To reflect past user interaction with the system, this episode prior is maintained as a vector of counts representing the number of times each of the $N + 1$ goals have been

completed in past episodes. Since g_0 is a goal unknown to the system, by definition, we cannot observe it or create a model for recognizing it. Thus, we use a proxy to count the number of times a non-highlighting goal has been observed and use that to empirically update the probability of g_0 . The proxy is defined by counting the number of times a typing goal has been observed using the typing events listed in Section 5.2.2. In effect, this gives an equal weighting to counting observed typing goals and observed highlighting goals. Note that this need not be the case; g_0 could be weighted differently (lower, for instance) to reflect beliefs about the chance of it occurring in a given setting. With these empirical counts, we also apply an add-one smoothing to avoid have goals with zero probability. Finally, this vector is normalized to obtain the updated episode prior distribution.

Note that as the number of observed goals, N , increases, the implementation for maintaining all the goal distributions will need to dynamically expand as well.

5.4 Simulation Experiments

In this section, we design simulation experiments to evaluate both the feasibility of using a goal model in real time, and the performance of the suggestion quality made using the goal model. To assess the suggestion quality, we situate the goal model within a help application for carrying out highlighting tasks while authoring slides. The assistance offered by this help application is the ability to suggest J highlighting macros via a pop-up toolbar. Section 5.4.1 describes the architecture of this system and the role the goal model plays.

Our main objective in this evaluation is to assess the impact of the goal model in an intelligent help system. Specifically, we assess the performance of a help system that uses information from a goal model in comparison to the same system without that information. Likewise, we compare the performance of the help system with the goal

model when it is equipped with the user characteristics model. The specific decision making policies we use in this comparative evaluation are defined in Section 5.4.3. As well, the metrics used to evaluate the goal model in comparison to other decision policies are described in Section 5.4.4.

In Section 5.4.2, we describe a simple simulated user model for generating events that interacts with our help application. With these experiment conditions, we present the results of the simulations in Section 5.4.5.

5.4.1 System Set-Up

Following the general DAISI system architecture presented in Figure 3.3, we focus on evaluating the User Goal Component in this experiment. This component takes an event as input and returns an updated belief distribution over user goals, corresponding to the steps described in Sections 5.2 and 5.3 of this chapter. To provide the context of the experiment, the rest of this section describes the remaining components identified in this architecture.

5.4.1.1 The User Characteristics Component

Chapter 4 detailed the development of user models in intelligent help applications. Here, we use a simple user model, focusing only on one trait, *tendency to need help* (TN), and one characteristic, *neediness* (NS). Figure 5.18 shows the DBN model we use to infer the user’s neediness and tendency to need help in this simulation experiment. We include the detailed parameters in Appendix B.2, and describe their high level definitions here.

At a high-level, the system assumes the user carries out one goal, G , in each episode. Therefore, G is modeled as a static variable in the DBN. This user goal influences which set of machine states the system is currently in. The set of machine states, ms_t , is defined by the set of finite automata states representing the goal instances as described in Section 5.2. In the DBN, we have $Pr(MS_0|G)$, which is the distribution of machine

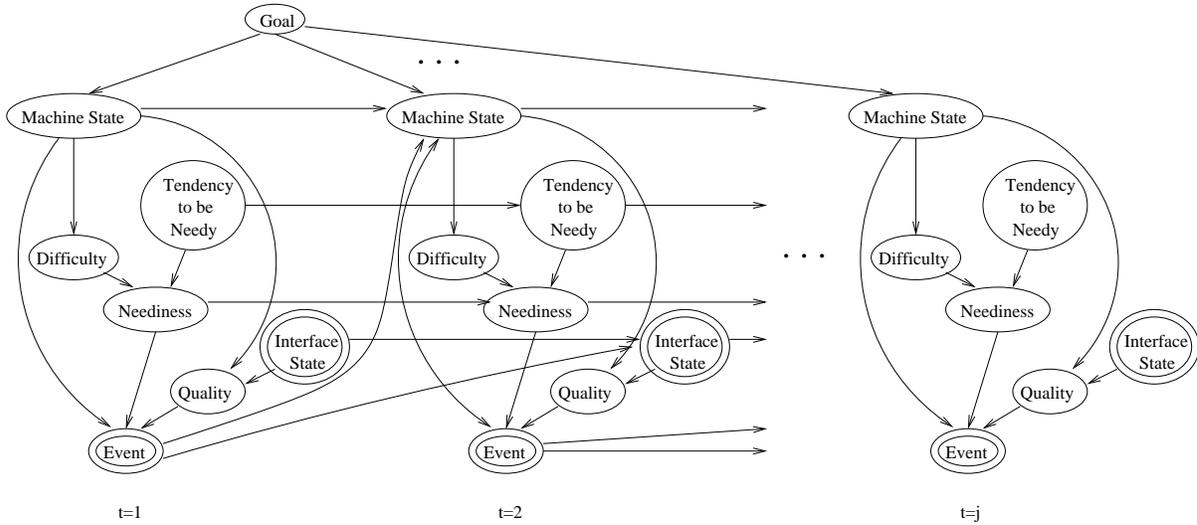


Figure 5.18: The simulated user DBN, showing one episode with j time steps.

states given the goal at $t = 0$, and $Pr(MS_t|G, MS_{t-1}, EV_{t-1})$, which defines the transition dynamics of machine states given the goal, the previous machine state, and the previous event, as defined according to the finite state model in Section 5.2. Corresponding to the start state, s_0 , in a given goal machine, we have $Pr(MS_0 = s_0|G) = 1$ for that goal.

To model a user's neediness level as a function of the machine state, we introduce a variable *difficulty* (DY_t) that abstracts the various machine states into difficulty levels by $Pr(DY_t|MS_t)$. The intuition is that given the current machine state and the target goal, DY_t captures the effort needed to complete that goal manually. In this simulation, DY_t takes values: easy (little effort required), medium, or hard (much effort required). Moreover, the definition is probabilistic so that when the current machine state is far away from the goal state, there is a greater chance of the user perceiving the difficulty to be hard, i.e., $Pr(DY_t = hard|MS_t) = 0.6$, a lesser chance of the user perceiving the difficulty is medium, i.e., $Pr(DY_t = medium|MS_t) = 0.3$, and a very small chance of the user perceiving the difficulty is low, i.e., $Pr(DY_t = easy|MS_t) = 0.1$. Defining $Pr(DY_t|MS_t)$ for other cases is done in a similar manner.

Although Figure 5.18 shows multiple copies of TN for convenience in the illustration,

it is defined as a static variable. The conditional probability tables for $Pr(NS_0|TN, DY_0)$ and $Pr(NS_t|TN, DY_t, NS_{t-1})$ are both handcrafted to reflect the following intuitions: a user is more likely to be needy when the task is hard, and less likely to be needy when the task is easy; the user who has a tendency to need help will be more likely to be in a needy state than a user who does not have a tendency to need help; a user who was needy at the previous time step will be more likely to remain needy at the current time step, and more so if the user has a tendency to need help or if the task difficulty is hard.

The interface state variable IS has four values that indicate whether system help is currently available or not, and if so, if the system’s estimated utility of help is low, medium, or high. Having three categories to capture the system’s estimated utility is a coarse-grained way of discretizing the utility value. Note that IS is not a random variable as it reflects the system’s action. In addition, if the previous event indicates the user is considering help, then the same suggestion remains available.

Given IS_t and MS_t , the system defines the quality of the suggestion using the variable $QUAL_t$. Analogous to IS_t , this variable is coarsely discretized into four values: not applicable, low, medium, high. When help is not available, quality is also not applicable. Otherwise, $Pr(QUAL_t|MS_t, IS_t)$ is defined as a function of the current machine state and the system’s (discretized) estimated help utility. For example, a machine state indicating that the user still needs to exert lots of effort in order to complete the goal is most likely of higher help quality, because the system is more likely to help the user reduce effort. Moreover, if IS_t is high, then $QUAL_t$ has a greater chance of higher quality than if IS_t is medium and low. On the other hand, a machine state indicating that the user is almost at goal completion is most likely of lower help quality, because the system help that could be offered is limited by the effort remaining to get to goal completion. Again, the actual value of IS_t will further modify the distribution of $QUAL_t$.

Finally, the DBN has an event model defined by $Pr(EV_t|MS_t, NS_t, QUAL_t)$. Here, EV_t is defined by the event vocabulary presented in Section 5.2.2. The main restriction

encoded is when $QUAL_t$ is not available, there is no system help, so EV_t cannot be an event that considers help or accepts help. The rest of the event model is handcrafted based on the following intuitions: select and font events generally do not correspond to the user’s neediness level in this set-up because our simulation mimics the user carrying out goal-oriented tasks in a controlled setting; undoing extraneous events in extraneous states indicates the user is less likely to be needy; undoing goal-related events in extraneous states indicates the user is more likely to be needy; other undo and behavioural events indicate the user is more likely to be needy; considering and accepting low quality help is more likely to indicate the user is needy.

5.4.1.2 The Action Selection Component

The design of this action selection component — the definition of suggestion utility, the greedy approximation to computing joint expected utility of system actions, the system’s myopic decision policy — follows closely that of the first case study in Chapter 4 presented in Section 4.3.2. Due to the changes in the application context and the notation used, we review the details below.

First, we describe the step in the action selection component that creates a help action corresponding to a toolbar of highlighting icons to suggest to the user. Recall that the library has a set of N goals, g_1, \dots, g_N , each with associated probabilities of being the current user goal p_1, \dots, p_N stored in the goal distribution. To express the usefulness of suggesting an icon to help the user with the true goal, we define a utility function $U(m_i|g_k, m.s_k)$, where m_i is the macro corresponding to goal g_i ($1 \leq i \leq N$), g_k ($1 \leq k \leq N$) is the true goal, $m.s_k$ ($1 \leq k \leq N$) is the current automaton state in g_k , and the utility is defined as the number of shared font features between g_i and g_k minus the number of different font features between them and minus the common features that have already been achieved between $m.s_k$ and g_i . Using the information given by the goal distribution, we define the expected utility of suggesting m_i as: $EU(m_i) = \sum_{k=1}^N U(m_i|g_k, m.s_k)p_k$.

Finally, to choose a set of goals to suggest, we define the joint expected utility of a set of J macros as:

$$JEU(m_1, \dots, m_J) = \sum_{k=1}^N \arg \max_{m_i: i \leq J} U(m_i | g_k, ms_k) p_k. \quad (5.2)$$

To speed up the calculation in the system, joint expected utility is implemented in a greedy fashion. Specifically, among the appropriate macros for each of the N possible goals, we choose the macro with the highest expected utility, $m_1 = \arg \max_{m_i} EU(m_i)$. Next, with $N - 1$ macros remaining, we select the macro that jointly makes up the best suggestion given m_1 , i.e., $m_2 = \arg \max_{m_2} JEU(m_1, m_2)$. The remaining $J - 2$ macros are chosen following this pattern to make up the set of macros to suggest to the user in a toolbar. This toolbar represents (a greedy approximation of) the best help action the system can offer to the user given the library of goals and the goal distribution.

To decide whether the help action should be offered to the user, we turn to the decision policy used in the action selection component. First, we consider a policy that uses only the goal model information as we have already described. Here, the expected utility of the toolbar action, A , can be defined as

$$EU(A) = EU(A|EV = acc)Pr(EV = acc) + EU(A|EV = \neg acc)Pr(EV = \neg acc), \quad (5.3)$$

where acc denotes the set of accept events defined in the event vocabulary and $\neg acc$ denotes all other events. In this equation, $EU(A|acc)$ is simply the joint expected utility value defined in Equation (5.2), $EU(A|EV = \neg acc)$ is defined as a small “cognitive” interruption cost of suggesting a toolbar that gets ignored, $Pr(EV = acc)$ is the probability that the user accepts A by summing up the individual probabilities of each accept event in the vocabulary, and $Pr(EV = \neg acc) = 1 - Pr(EV = acc)$. Following our previous work described in Chapter 4, $Pr(EV = acc)$ is calculated the same way as described in Equation (4.3) of Section 4.3.1 by using clique tree inference on the Bayesian user model. When no help is offered, the utility is 0. Thus, this policy suggests the action

A if $EU(A) > 0$. Since this policy makes use of information in the goal model only, we refer to this policy as the goal model policy, denoted as **GM**.

Next, we define a policy that makes use of information from both the goal model and the characteristics model, which is denoted as **GC**. Policy GC also defines the expected utility of the help action as in Equation (5.3). However, $EU(A|EV = acc)$ and $EU(A|EV = \neg acc)$ are defined using expectation with respect to the user's neediness level. Specifically, we use the following definitions:

$$EU(A|EV = acc) = \sum_{NS} R(JEU, NS)Pr(NS) \quad (5.4)$$

and

$$EU(A|EV = \neg acc) = \sum_{NS} C(NS)Pr(NS) \quad (5.5)$$

where JEU is the value computed in Equation (5.2), NS is the user's neediness level, R is the reward function, C is the cost function, and $Pr(NS)$ is the characteristics distribution updated by the DBN model in Figure 5.18. Note that the reward function takes both JEU and the current neediness level of the user as parameters. Thus, for the same value of assistance, users who are more needy will find that help more rewarding while less needy users will find the help less rewarding. Although the cost function prescribes an interruption cost associated with each system action regardless of the value of the assistance, this cost is also defined as a function of the user's neediness level. In the current implementation, highly needy users will find the given help twice as valuable with an associated unit interruption cost, users who are not needy will find the given help half as useful and twice as costly, while the reward and cost for users who are somewhat needy are defined with midpoint values of the other two neediness settings. (The detailed parameters for R and C are provided in Appendix B.2.) Similar to the GM policy, this policy suggests the action A if $EU(A) > 0$.

5.4.2 Simulated User for Executing Highlighting Goals

We create a simple simulation model to interact with the help system in this experiment. Corresponding to the user model from Figure 5.18, we create two user types that define whether the user has the tendency to be needy ($TN = \text{true}$) or not ($TN = \text{false}$). At the start of each trial, the simulated user model is equipped with a set of N randomly generated highlighting goals. For ease of comparison, rather than allowing these goals to vary in the number of font features, we fix $K = 4$.

Each trial consists of a series of episodes, where each episode marks the completion of a goal. At the start of each episode, a new goal is sampled. Each of the N goals have a predetermined prior probability of being drawn during a trial. For N goals with N distinct indices, we define the prior distribution as a function of the goal indices, so that goals with larger indices have a higher chance of being sampled.

The event generation model of this simulated user is designed to mimic the scenario of a controlled user study where users carry out a series of highlighting tasks in a help system (which we conduct and report in Section 5.5). Given a goal, the simulated user will generally execute events to complete that goal. However, if the user type is needy, then a random behavioural event will be generated at regular intervals to indicate the user needs help. If the system offers help, the simulated user whose type is needy will consider the help 90% of the time, while the simulated user whose type is not needy will only consider help 30% of the time. Once help is considered, a needy user will accept the best help available, while a non-needy user will only accept help that matches the target goal exactly. This process repeats until the goal is completed. When the episode ends, a new goal is sampled and this process repeats. Note that this event generation model is quite restrictive; it does not generate undo events to simulate backtracking behaviour. Nonetheless, this set-up will serve as an initial assessment of the speed and value of the model. In Section 5.5, we present a more realistic evaluation to assess the utility of the goal model component with real users.

5.4.3 Comparison System Policies

In Section 5.4.1, we described the system’s action selection component and the use of two possible policies, GM and GC. For comparison purposes, we add the following three policies in our simulation experiment:

1. Never help, **NH**: Regardless of the input information into the action selection component, this policy never offers any help to the user under any circumstances.
2. Frequency model, **FM**: A system using the frequency model may learn a distribution over goals purely based on the number of times observed goals have been executed in the interaction. We define a policy that only uses this information to always suggest help by offering a toolbar consisting of the most frequently occurring goals. Specifically, at each time step, when the library is not empty (i.e., $N > 0$), the system will suggest a toolbar with $\min(J, N)$ icons. After creating the toolbar of macros, we define the reward of this toolbar as the utility afforded by the macro that maximizes the expected utility with respect to the goal frequency distribution:

$$MES = \max_{m_i} \sum_{k=1}^N U(m_i | g_k, ms_k) p_k \quad (5.6)$$

Here, m_i is a macro considered for suggestion, g_k is a goal in the library, ms_k is the current automaton state in g_k ’s machine, p_k is the corresponding goal’s frequency, and U is defined as in the GM and GC policies. In Equation (5.6), we use maximization because we expect users to select the best option available (although this is not necessarily always the case with real users). Each system action also has an associated cost. We use a small, unit cost to represent the cost of interruption of each toolbar here (same as the cost unit for the GM policy).

3. Frequency model and characteristics model, **FC**: Analogous to the GC policy, we supplement the frequency model using the characteristics distribution to create

this policy. First, the system will create the best action possible using the frequency model. At this point, we have a potential suggestion with $\min(J, N)$ icons. Adopting Equation (5.3) and Equation (5.4), we calculate the expected utility of suggesting an action to the user. However, we do not use joint expected utility in this policy. Instead, the term $EU(A|EV = acc)$ is defined using MES from Equation (5.6) as follows:

$$EU(A|EV = acc) = \sum_{NS} R(MES, NS)Pr(NS) \quad (5.7)$$

where all other terms are defined the same as in Equation (5.4). Finally, when no help is offered, the utility is 0. Thus, this policy suggests the action A if $EU(A) > 0$.

All the policies above make use of an action parameter J that determines the maximum number of icons that the system may offer to the user in a toolbar. As discussed in the results below in Section 5.4.5, we set $J = 1$ and $J = 3$ for comparison.

5.4.4 Evaluation Metrics

The measures we use to evaluate the system's performance are speed, accuracy, and suggestion quality. In terms of speed, we will measure both computational requirements for the goal model and for the characteristics model because the performance of both models is dependent on the number of goals observed throughout the interaction. For accuracy, we will measure user goal inference accuracy based on the goal distribution and user type inference accuracy based on the characteristics distribution. In terms of suggestion quality, we will measure objective quality based on event execution as well as subjective quality as defined by the system's utility function. Specifically, the metrics we use in this experiment are:

- computational speed:

- **goal model inference time** — at each time step, the time taken for the system to update the goal distribution incrementally, averaged over episodes and trials
- **characteristics model creation time** — at the end of each episode, the time taken for the system to create a new and potentially larger DBN as a result of newly added observed goals, averaged over episodes and trials
- **characteristics model inference time** — at each time step, the time taken for the system to update the characteristics distribution, averaged over episodes and trials
- inference accuracy:
 - **user goal inference accuracy** — for each system suggestion, the goal distribution’s probability of the true goal, averaged over episodes and trials
 - **user type inference accuracy** — at each time step, the characteristics distribution’s probability of the true user type averaged over trials
- objective suggestion quality:
 - **task completion effort** — for each goal, the number of entry, progress, and response events required to execute each goal (i.e., all the events excluding behavioural events), averaged over episodes and trials
- subjective suggestion quality:
 - **utility of suggestion** — for each system suggestion, the utility of the accepted help as defined by the system’s utility function in expectation of the inferred user state, averaged over trials

5.4.5 Results

The simulation results presented in this section are run on Linux with an Intel(R) Xeon(R) CPU 5150 processor at 2.66GHz CPU and with 8GB memory. In the results below, we will choose different settings using the parameters J and N . When N is small, e.g., $N = 5$, those simulation trials will mimic the scenarios where users want to execute a series of highlighting goals in a slide presentation, but the goals are often the same in order to preserve aesthetic consistency. When N is large, e.g., $N = 15$, those trials will mimic users creating slides with higher aesthetic variance.

5.4.5.1 Computational Speed

Although it is rare that users apply a large variety of highlighting goals in a presentation, we use the setting of $N = 15$ unique goals in each trial in this simulation in order to observe the speed performance with increasingly large models. For each combination of user type and policy, we ran 100 simulation trials, each with 100 episodes. We use a large number of episodes in order to improve the chance of observing all N goals being sampled in each trial.

The first result we present is the time required for goal inference in Figure 5.19. Recall that our goal model makes use of an incremental inference procedure documented in Section 5.3. The timing results here are based on the policies GM and GC, and are averaged over 200 trials. As expected, this procedure is efficient, and scales linearly in N .

Next, Figure 5.20 shows the computational speed for dynamically creating the user model and for exact clique tree inference on the DBN from Figure 5.18. The timing results here are based on the policies FC and GC, and are averaged over 200 trials.

Recall that model creation is needed as new goals are added to the system's library. Since the set of machine states are represented as part of the DBN, the DBN also grows whenever a new goal is added to the library. Currently, this addition takes place at the

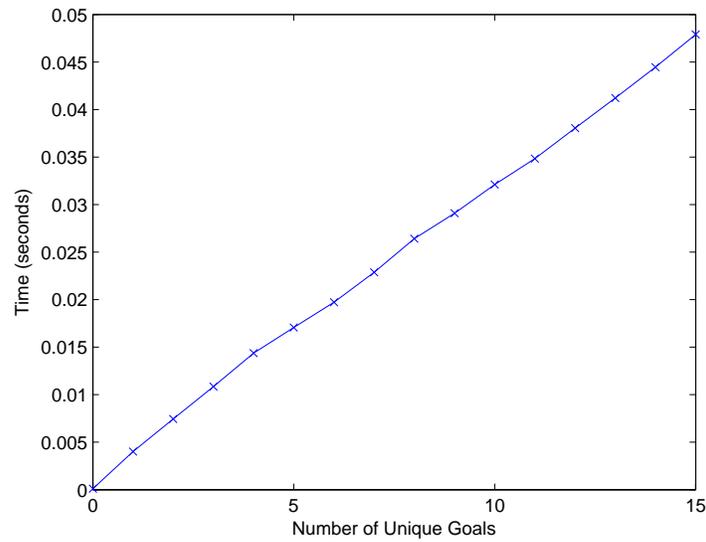


Figure 5.19: Computational speed for goal model inference time.

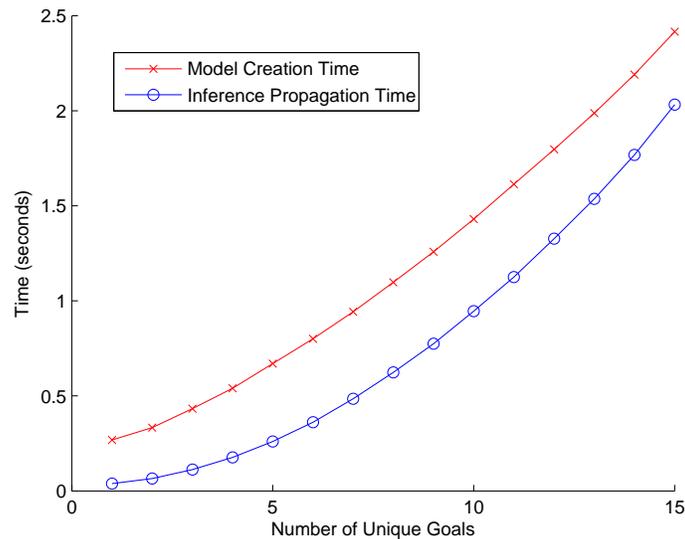


Figure 5.20: Computational speed for characteristics model creation time (top) and characteristics model inference time (bottom).

end of each episode (*if* a new goal is observed). We see in Figure 5.20 that model creation time requirements are manageable (starts off at approximately 0.27 seconds with $N = 1$).

When $N = 4$, the required model creation time becomes greater than 0.5 second, and the time becomes greater than 1 second when $N > 7$. In a controlled experiment (such as a simulation environment or a human study with predetermined goals), dynamical model creation time can be avoided by creating a user model that includes all the predetermined goals upfront, although this solution results in slower model inference time at each time step. In real world environments, possible methods to speed up this process include using a separate processor to update the model or updating the model dynamically only when the user is inactive. It is also important to note that unlike online goal inference, model creation can take place asynchronously using a “lagging” model. Specifically, when the user characteristics model needs to be updated, it can be done in a separate process, while the current characteristics model is still used in the system’s reasoning module. While this setup uses a characteristics model that is slightly outdated, we suspect that overall system performance to not be affected as we do not expect these characteristics to change that quickly.

From Figure 5.20, we also see that exact clique tree inference that takes place at every time step is fast when N is small (approximately 0.0388 seconds at $N = 1$, and increases to over 1 second for $N > 10$). Since this result is based on the use of exact inference, an obvious solution to speeding up this process is to use approximate inference [BK98]. Alternatively, separating the inference procedure to run on another processor is also a possibility. Here, inference time is more critical and the “lagging” model is not applicable.

5.4.5.2 Inference Accuracy

In this experiment, we set all the goals to have $K = 4$. In other words, including the event for selecting a string, if we were to manually execute one of these goals (without any negative progress or behavioural events), five events would be required to accomplish these goals. We ran 100 trials, each for 100 episodes. Overall, we would like to know

the probability of the true goal as each event is executed. We conducted this experiment using the GM policy and a simulated user that does not accept any help at all. At each time step, we monitored the probability of the true goal. The graphs in Figure 5.21 show the accuracy results for goal inference with a library size of $N = 5$.

This figure is divided into five graphs. Reading downwards, each graph corresponds to the probability of the true goal, g^* , as a function of an increasing number of events being observed; the top graph plots $Pr(G|SEL)$, the second plots $Pr(G|SEL, EV_1)$, the third plots $Pr(G|SEL, EV_1, EV_2)$, and so on, with EV_1 to EV_4 indicating the K font events of g^* . Viewing the graph in this manner lets us see the change in the system's belief of the true goal over the course of an episode. Reading sideways from left to right, the plots indicate the probability of g^* (given the corresponding number of events observed) as a function of the number of times that goal has been previously achieved in the same trial. This dimension plays a role in the inference accuracy because a goal that has been successfully completed more often in the past (via the same sequence of events) will strengthen the counts used to empirically update the episode prior and the observation model, so the goal will be considered more likely than if fewer occurrences of it had been observed previously. However, this factor is not completely controlled in this experiment because there may be other goals that have been observed just as many time as the one of interest. Thus, as the number of times a goal was observed increases, we would generally expect the accuracy to improve.

We discuss Figure 5.21 by inspecting the graphs downwards. (Note the difference in the y-axis scales.) At the beginning of the episode when only the initial select event has been observed, the true goal becomes more and more likely as more occurrences of it have been seen in the past interaction. As explained above, this probability increases over time because the episode prior distribution is empirically updated. Next, we turn to the second graph in Figure 5.21. After observing the first font event of the goal, the system's belief that this is the true goal approximately doubles from its belief in the

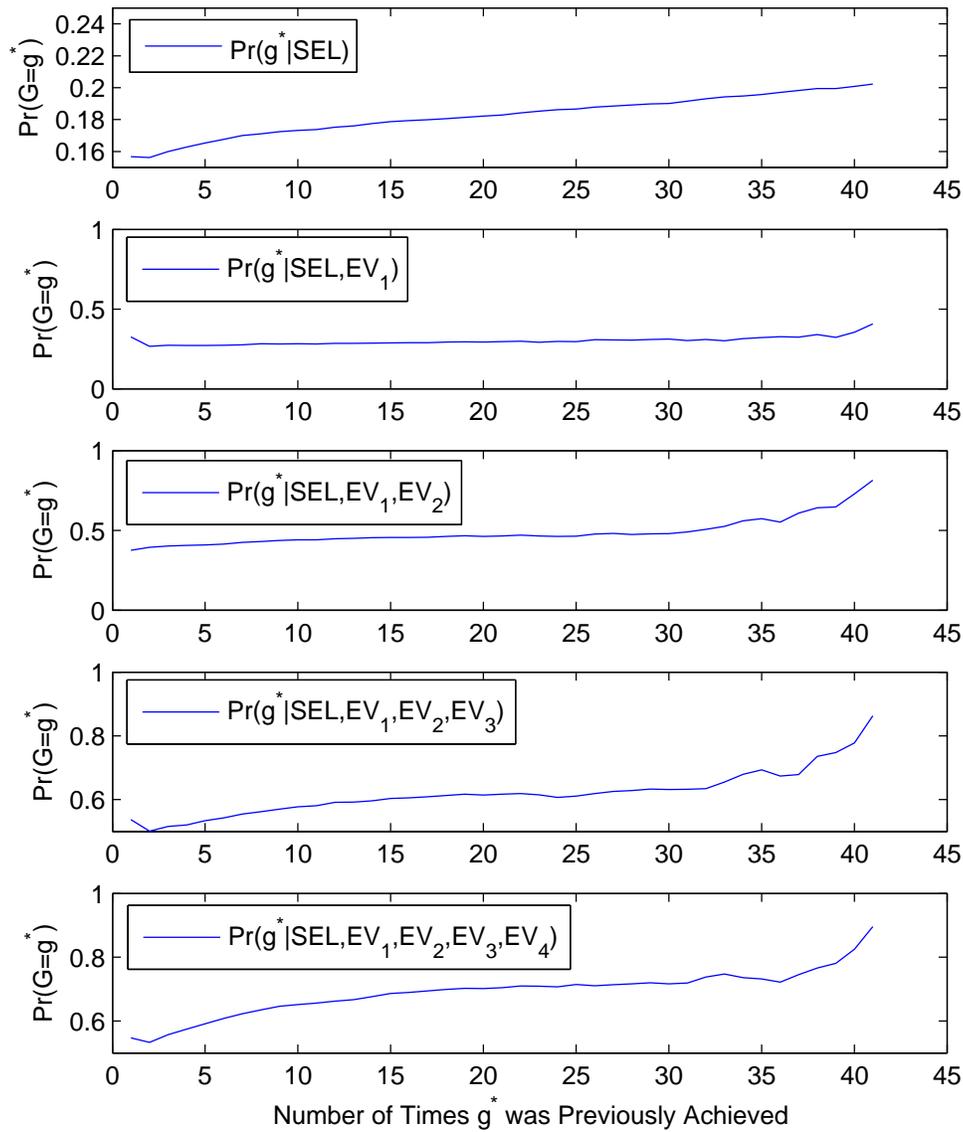


Figure 5.21: Goal inference results showing the probability of the true goal as a function of the number of times it was observed within a trial. Note the difference in scale of the y-axes.

previous time step when only select was observed. Similarly, the third graph shows that the system’s belief increases with an added observation of a font event, although the

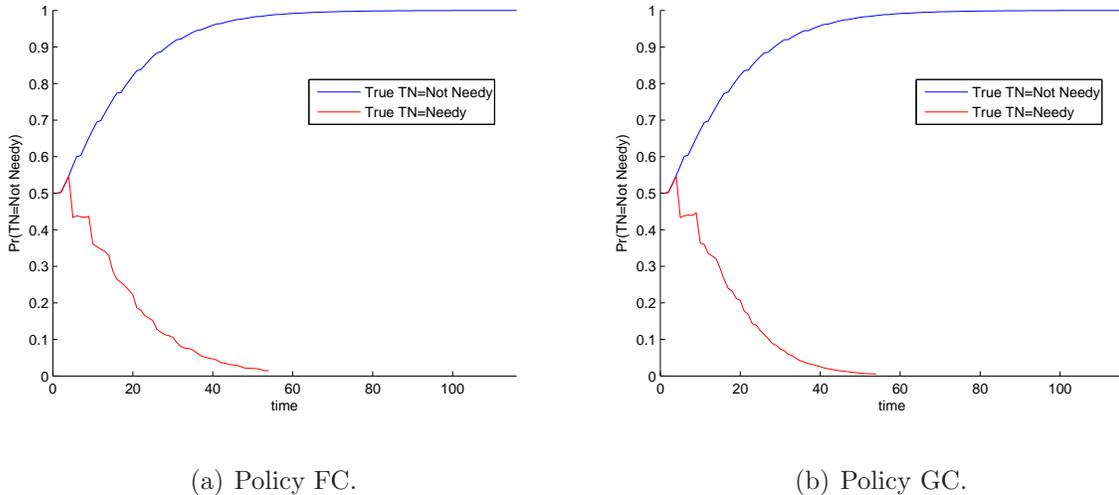


Figure 5.22: Characteristics inference results for $Pr(TN)$ using $J = 1, N = 5$.

increase is smaller in this case. The last two graphs are interpreted similarly. Note that in general, after the system has observed two font events from the goal, the probability of the true goal is more than 0.5. Interpreting the graph in this manner, we see that the belief of the true goal generally improves as the system has more experience with those goals.

Next, we turn to the accuracy results for inferring the user’s true type. Among the five policies of interest, only two make use of the characteristics model. Thus, we ran the experiment using policies FC and GC for 100 trials, with 25 episodes each, using the setting of $J = 1$ and $N = 5$. Figure 5.22(a) shows the results for inferring the user type under the policy FC, while Figure 5.22(b) shows the corresponding inference results under the policy GC. Note that the data indicates there are fewer time steps involved for the needy cases. The reason is that needy users accept help more, which reduces the overall number of events executed by those users. (Recall that we do not model error correction on the part of the user.) From these results, we see that in both cases, the system is able to infer the true user type accurately and fairly quickly.

The ability for the system to infer the true user type varies according to how much help is offered to and accepted by the simulated user. In particular, we show an example

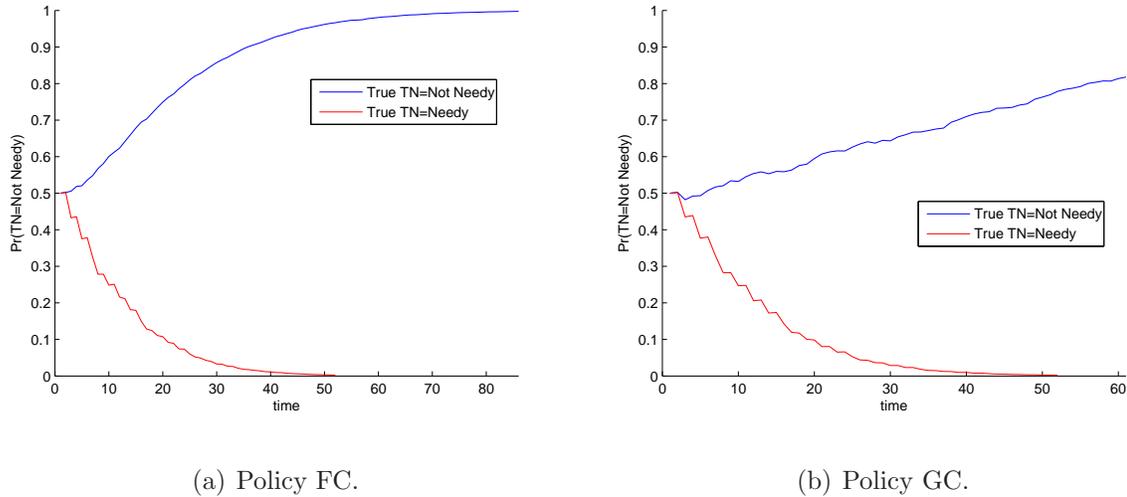


Figure 5.23: Characteristics inference results for $Pr(TN)$ using $J = 3, N = 10$.

simulation result using the setting $J = 3$ and $N = 10$ with policies FC and GC for 100 trials with 25 episodes each in Figure 5.23. We see that the system takes longer to infer the type when the user is not needy, especially when GC is used. As we see below, this result is due to the increased amount of help that is offered to and accepted by the simulated users in this setting.

5.4.5.3 Task Completion Effort

Recall that the policy NH never helps the user, so that the effort measured in this experiment reports the average “manual” effort required by the simulated user for baseline comparison purposes. For this purpose, we do not count behavioural events. On the other hand, the remaining four policies suggest help with J icons. In the case of these adaptive policies, we count the execution of accept events as part of the task completion effort. Note that when our simulated user accepts help, it is designed to select the icon that best matches the true goal. In order to compare the different policies based on how it chooses its action, we create settings where the number of goals in the library, N , is larger than the number of icons to be suggested, J . For comparison purposes, we fix $J = 3$ and use one of $N = 5, N = 10, \text{ and } N = 15$. In these settings, we compare the performance for

Table 5.10: Average amount of effort per episode based on the number of goal-related events with $J = 3$ and $N = 5$.

		NH	FM	FC	GM	GC
$TN = \text{true}$	min	5.00	2.00	2.00	2.00	2.00
	avg	5.00	2.01	2.03	2.22	2.00
	max	5.00	3.00	5.00	5.00	3.00
	stderr	0.00	0.09	0.30	0.78	0.05
$TN = \text{false}$	min	5.00	2.00	2.00	2.00	2.00
	avg	5.00	2.67	4.93	4.03	3.91
	max	5.00	6.00	5.00	5.00	5.00
	stderr	0.00	1.01	0.45	1.40	1.44

2 user types and 5 policies by running 100 trials with 25 episodes per trial.

By counting the events carried out related to completing the goal, such as selecting the string, applying font attributes, and accepting help, we calculate an approximation to the amount of effort spent in each episode on average. Since we set all the goals to $K = 4$, the maximum amount of effort possible is the execution of 6 events (e.g., the user manually executes the select event, four font events, then accept system suggestion). Note that this is a naive approximation because the simulated user may accept incorrect help. In a real world situation, when incorrect suggestions from the system are accepted by the user, the user would need to make changes to fix the goal pattern. We do not simulate correction behaviour in this experiment, and simply report on the effort expended up to the point when a goal is completed or when a suggestion is accepted only. Table 5.10 to Table 5.12 show the results for this experiment.

The effort reported under policy NH indicates the baseline of executing the goal without help. For the remaining policies, we see that the effort needed when $TN = \text{false}$

Table 5.11: Average amount of effort per episode based on the number of goal-related events with $J = 3$ and $N = 10$.

		NH	FM	FC	GM	GC
$TN = \text{true}$	min	5.00	2.00	2.00	2.00	2.00
	avg	5.00	2.01	2.05	2.18	2.02
	max	5.00	3.00	5.00	5.00	5.00
	stderr	0.00	0.11	0.35	0.71	0.20
$TN = \text{false}$	min	5.00	2.00	2.00	2.00	2.00
	avg	5.00	2.67	4.81	4.35	3.81
	max	5.00	6.00	5.00	5.00	5.00
	stderr	0.00	0.94	0.72	1.24	1.46

Table 5.12: Average amount of effort per episode based on the number of goal-related events with $J = 3$ and $N = 15$.

		NH	FM	FC	GM	GC
$TN = \text{true}$	min	5.00	2.00	2.00	2.00	2.00
	avg	5.00	2.01	2.13	2.22	2.05
	max	5.00	3.00	5.00	5.00	5.00
	stderr	0.00	0.09	0.54	0.77	0.37
$TN = \text{false}$	min	5.00	2.00	2.00	2.00	2.00
	avg	5.00	2.68	4.44	4.37	4.00
	max	5.00	6.00	5.00	5.00	5.00
	stderr	0.00	0.98	1.09	1.16	1.36

is higher than when $TN = \text{true}$ because users who are not needy generally consider and accept less help. Note that overall, the standard error is higher for non-needy users. For

needy users, all the policies perform similarly, with GM requiring a bit more effort than the other policies that offer help. For non-needy users, FM performs better than FC, GM, and GC because FM always offers help (regardless of the suggestion quality), so there are more opportunities for the simulated user to consider and accept help. As a result, FM reduces effort more than the other adaptive policies. Although we observe this gain in effort here, we will see that this same advantage yields lower performance for FM according to other evaluation metrics below.

Considering only FC, GM, and GC, we see that the average amount of effort improves slightly for non-needy users, going from FC, then GM, and then GC. This comparison suggests that the two goal-based policies offer better help for non-needy users.

5.4.5.4 Frequency and Utility of Suggestion

This section also reports on the comparison with 5 policies, 2 user types, and the settings using $J = 3$ help icons and predefined goal sets of size $N = 5$, $N = 10$, and $N = 15$. For each combination, we ran 100 trials, with 25 episodes per trial.

As part of measuring subjective utility of assistance, we report on the percentage of suggestions made by the system with respect to the total number of entry and progress events. (Note that when the user executes a behavioural event such as `pause` or `browse`, the system does not interrupt that behaviour. Likewise, when the user executes a response event such as `cons` or `acc(bold)`, the system does not attempt to make another suggestion.) Among the suggestions made, we also show the percentage of acceptances made by the simulated user in response. These numbers are shown in Table 5.13 for each case in the format of “Percent Accepted By User/Percent Suggested By System”.

Using FM as a reference, we see that needy users accept help almost all the time while not needy users accept help at most two-thirds of the time. For the adaptive policies FC, GM, and GC, we see a similar pattern for needy users: the system recognizes the opportunity to offer help the majority of the time and the simulated user is generally

Table 5.13: Average percentage of acceptances and suggestions per trial with $J = 3$ and one of the listed values of N . Results for NH are omitted since it does not offer suggestions.

	FM	FC	GM	GC
$N = 5$: $TN = \text{true}$	99.2/100.0	98.6/96.8	99.1/73.6	99.7/100.0
$TN = \text{false}$	59.5/100.0	16.1/1.5	55.1/3.9	54.2/21.8
$N = 10$: $TN = \text{true}$	98.9/100.0	98.2/95.7	98.9/77.3	99.4/98.5
$TN = \text{false}$	59.7/100.0	33.3/2.7	48.7/11.2	56.1/23.4
$N = 15$: $TN = \text{true}$	99.2/100.0	98.9/90.0	99.0/72.1	98.5/93.8
$TN = \text{false}$	57.3/100.0	38.8/3.0	41.2/5.1	42.1/16.3

receptive.

On the other hand, the system offers less help for users who are not needy. In particular, FC makes very few suggestions overall, as low as 1.5% of the time up to 3.0% of the time. The percentage of system suggestions increase slightly for GM, and increase even more for GC. Note that the percentage of user acceptances is only between 15% and 40% for FC, while the user acceptance percentages are generally higher for GM and GC, in the range of 40% and 57%. In particular, even though GC offers help more often than FC and GM, the percentage of user acceptance is about the same or higher than those of FC and GM. Overall, the results in Table 5.13 suggest that the various adaptive policies are able to respond to different user types to varying degrees.

For reference, we computed the objective utility of assistance by calculating the effort saved from each accepted suggestion in terms of the number of goal-related events, and report the results in Table 5.14. Recall our simulated user acceptance model — needy users accept 90% of the help offered (by choosing the best option among those suggested) and non-needy users accept only perfect help 30% of the time. While this acceptance

Table 5.14: Average objective effort (number of events) saved from accepting system help. Results for NH are omitted because it does not offer suggestions.

	FM	FC	GM	GC
$N = 5$: $TN = \text{true}$	2.5	2.5	2.8	2.5
$TN = \text{false}$	2.7	0.6	2.9	2.9
$N = 10$: $TN = \text{true}$	2.2	2.3	2.4	2.2
$TN = \text{false}$	2.2	1.1	2.1	2.6
$N = 15$: $TN = \text{true}$	1.9	2.0	2.1	1.9
$TN = \text{false}$	1.9	1.2	1.1	1.6

model is naive, it gives us an initial assessment of our model. (A user experiment is reported in the next section to provide a more realistic evaluation.) Table 5.14 reports the number of events saved in carrying out a highlighting goal in the simulated environment. For example, in the case of a needy user under the FM policy for $N = 5$, there is a savings of about 2.5 events on average, for highlighting goals that require 5 event execution (that is, the select event, and four font attribute events). Since highlighting help is only available after an entry event is completed, automated help can at best save the user the execution of 4 font events. In this case, the suggestions are helping the user with approximately 62.5% of that objective effort. On the other hand, a non-needy user under the FC policy for $N = 5$ only receives savings of about 0.6 events on average. Here, the system is saving the user with approximately 15% of the objective effort. Generally, we see that our naive simulated user acceptance model results in positive objective savings in accepted help.

With this in mind, we turn to the results for assessing the subjective utility of help in Table 5.15. The numbers reported here are calculated using the expected utility of suggestion as defined for each of the policies above, since these quantities consider the

Table 5.15: Average expected utility per suggestion with $J = 3$ and one of the listed values of N .

			NH	FM	FC	GM	GC
$N = 5:$	$TN = \text{true}$	avg	0.0	6.5	50.6	20.3	54.2
		stderr	0.0	1.2	8.0	1.2	0.4
	$TN = \text{false}$	avg	0.0	-32.9	-0.7	2.2	14.0
		stderr	0.0	10.5	1.6	5.0	17.4
$N = 10:$	$TN = \text{true}$	avg	0.0	6.1	49.1	21.0	55.6
		stderr	0.0	1.3	10.1	1.0	5.0
	$TN = \text{false}$	avg	0.0	-34.0	0.8	0.6	14.1
		stderr	0.0	12.2	3.4	4.3	13.2
$N = 15:$	$TN = \text{true}$	avg	0.0	4.7	46.3	18.5	53.4
		stderr	0.0	7.0	13.5	4.0	6.9
	$TN = \text{false}$	avg	0.0	-39.2	1.0	-0.4	8.6
		stderr	0.0	12.8	3.4	2.4	16.6

reward of accepting help and the cost of interruption, in expectation of the (estimated) neediness level of the user at the time of the suggestion.

Using NH as the baseline reference, we see from Table 5.15 that offering too much help or the wrong kind of help induces overall negative utility for users, especially those who are not needy. Note that overall, the standard error is higher for non-needy users. In the case of FM where suggestions are made all the time, a negative utility is induced. In comparison, by offering fewer suggestions to non-needy users, GM is able to generally accumulate positive utility and thus, not annoy this type of user.

When user characteristics are used, FC outperforms FM — it is able to make better suggestions for needy users and annoy non-needy users less often. At the same time,

FC does better than GM overall — FC obtains more positive expected utility for needy users and in two of the three cases, FC accumulates less negative expected utility for non-needy users.

When the goal model and the user characteristics models are used in combination, GC performs better than all the other policies. Like FC, GC outperforms FM because it is able to accommodate user types. In comparison to GM, GC makes more suggestions for needy users, and thus, accumulates higher expected utility. Since Table 5.10 to Table 5.12 and Table 5.14 show that non-needy users using GC expend less effort than non-needy users using the GM policy, this suggests that GC generally offers better quality help for non-needy users than GM. Finally, GC outperforms FC in all the cases.

5.5 Usability Experiments

The purpose of this experiment is to compare the simulation results from Section 5.4 with those of human users. The simulation results above suggest that policies GM and GC perform better than policy NH that does not help the user, and also perform better than the adaptive policies FM and FC which are not decision-theoretic. This user study replicates the comparison by asking users to carry out highlighting tasks using policies NH, FM, GM, and GC. Details of the experiment set-up and outcomes are reported below.

5.5.1 Experiment Set-Up

The testbed application used in this experiment is PowerPoint 2003. Users were asked to complete the authoring of a slide presentation with a total of twenty-five slides by carrying out one highlighting task in each slide. For better experimental control, each slide has a target goal displayed in the title of the slide, a sentence in the body of the slide typed in the default text pattern, images, and a slide number at the bottom. Each task

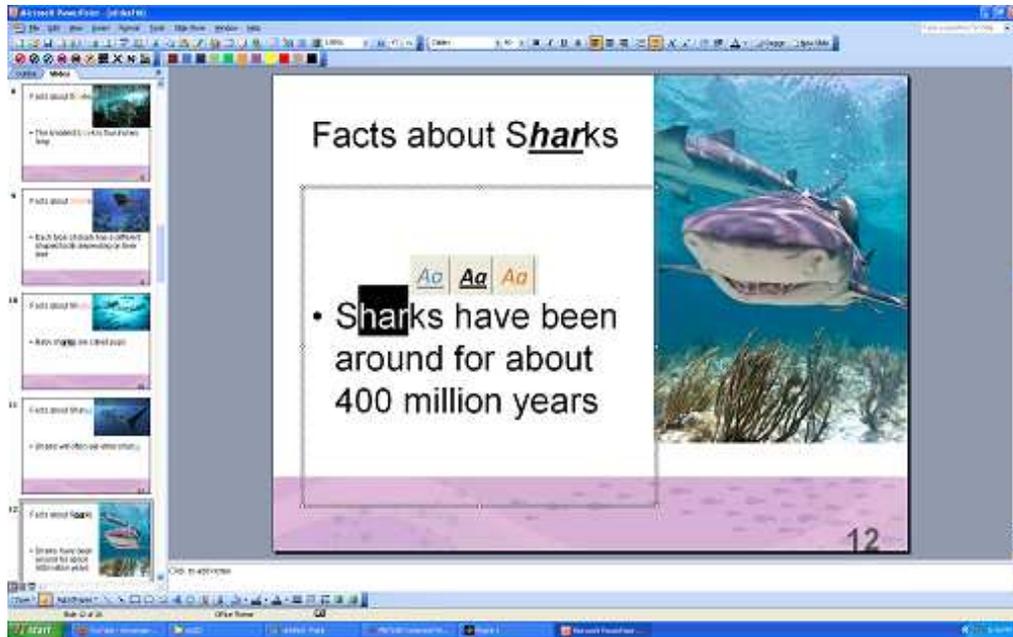


Figure 5.24: Macro suggestions with highlighting icons implemented in PowerPoint 2003.

requires the user to replicate the title’s highlighting pattern in the main sentence of the slide. Figure 5.24 shows an example slide with a toolbar of three icons, each suggesting a particular highlighting pattern that can be applied to the selected letters “har.”

Although it did not play a role in the simulation experiment, the ordering of the macros in the toolbar will likely affect the user’s perceived utility of the suggestions. Thus, to be fair across the different comparison conditions, all the adaptive policies are implemented to present their macros in the order of highest to lowest value, where the macro with the highest value is placed closest to the mouse focus. Here, value is defined according to the specific comparison policy. For example, FM will use frequency to determine highest value while GM will use utility in expectation of the goal distribution.

We designed a within-subject experiment where one participant undergoes multiple test conditions and compare the different systems at the end in a post-questionnaire. The main constraint in these designs is the participant’s mental capacity for undergoing several conditions in one session. Thus, we limit the duration of each session to 90

minutes and restrict the number of test conditions to four. As a result, the test policies in this experiment are NH, FM, GM, and GC. In contrast to the simulation experiment, we eliminated the comparison of FC from this study. Among these policies, NH serves as a baseline policy.

In this study, we are particularly interested in comparing the performance of GM to that of FM in order to assess the utility contribution of the distributional formulation of goals in the Goal Model Component. In other words, our analysis will compare the mean participant measures (e.g., manual effort, task completion time, subjective questionnaire responses) in the FM condition and test whether the corresponding mean participant measures in the GM condition are the same. Secondly, we are also interested in comparing the performance of GC to that of GM in order to assess the utility contributions of the User Characteristics Component. In this case, we compares the mean participant measures in the GM condition and test whether the corresponding mean participant measures in the GC condition are the same.

At the beginning of each session, participants are trained using the system without any help and with adaptive help so that they can become familiar with the task and interface. Since all of our participants have used PowerPoint before, the training session generally lasted for 5 minutes on average. (Note that the participants' PowerPoint experience was not controlled for in this study.) Using each of the four test policies, participants were asked to complete a presentation with 25 slides by carrying out one highlighting goal on each slide. The presentation slides used in each condition have different content but the same goals and goal distributions are used. In particular, the highlighting goals used and the frequencies of each goal were as follows:

- Orange, Bold, Underline, Shadow – 6 occurrences
- Blue, Underline, Italics – 6 occurrences
- Pink, Bold, Underline – 5 occurrences

- Bold, Underline, Italics – 4 occurrences
- Blue, Bold, Shadow – 2 occurrences
- Orange, Italics – 2 occurrences

The complexity and the number of goals were chosen based on the speed limitation of our inference engine (as reported in Section 5.4.5) and the desire to have goals with varying frequencies. These goals were randomly distributed throughout the slides in the presentations. Although the same goals are used in all four conditions, we minimized the potential learning effect by counter-balancing the order of the conditions across participants in the experiment. For all adaptive policies, we set $J = 3$. Since the icons suggested in the adaptive policies vary in their order of presentation, we expect interface learning effects to be minimal.

A small pilot study with 4 participants was conducted to test the feasibility of the experiment set-up. Based on informal feedback from these participants, the system's utility function in the GC and GM conditions were tweaked to be more sensitive to users who prefer no help. (Recall from Section 5.4.5 that all the adaptive policies made suggestions very often.) In addition, we found that participants often completed highlighting goals partially, then switched focus to view if the pattern matched the target pattern in the title of the slide. If not, then they would re-select the phrase and continue the highlighting task. Since our goal model treats every select event as the start of a highlighting episode, our system viewed this behaviour as users trying to complete multiple goals. This interpretation led to the system storing partially completed goals as new goals, and causing slower system performance nearing the end of the presentation. To handle this problem, we augmented the goal recognition model so that it only recognizes one goal per slide by storing the most recently completed goal. To be fair across all the conditions, we ran the characteristics inference engine in the background in all the cases even if its output is not used. These changes are a result of the limitation imposed by

Table 5.16: Average number of events executed by policy.

	NH	FM	GM	GC
avg	4.41	3.84	3.45	3.48
stderr	0.33	0.52	0.62	0.47

the speed issues of the characteristics inference engine. In general, approximate inference should be used to handle this problem. Overall, we collected data from 12 participants and report their results below.

5.5.2 Results

First, we report on the manual effort required in each condition. Table 5.16 shows the number of events executed by policy, averaged over all the participants and tasks. Overall, we see that NH requires the greatest number of manual event executions, followed by the adaptive policies. The difference observed here suggests that accepted automated help reduces the manual effort required in carrying out highlighting tasks. No significant differences were found among the adaptive policies.

As another objective measure, we investigated the task completion times in each condition. Table 5.17 reports the average times by policy, and Figure 5.25 shows the individual times for each participant and each policy, averaged over tasks. Although we see that GM and GC are slightly faster than NH and FM, there is no statistically significant difference found based on our t -test analysis. (Again, we assume independence, normality, and homogeneity of variances. These assumptions are ideally tested with a large, random population sample. As such, our results here are only suggestive.) Also, NH did not result in the slowest task completion times for all the participants as one might expect — automated suggestions may cause participants to accept incorrect help, thus, resulting in more time to correct the highlighting pattern, or may cause participants

Table 5.17: Average task completion times by policy (in seconds).

	NH	FM	GM	GC
avg	10.75s	11.19s	9.99s	10.45s
stderr	3.43	2.61	2.74	2.90

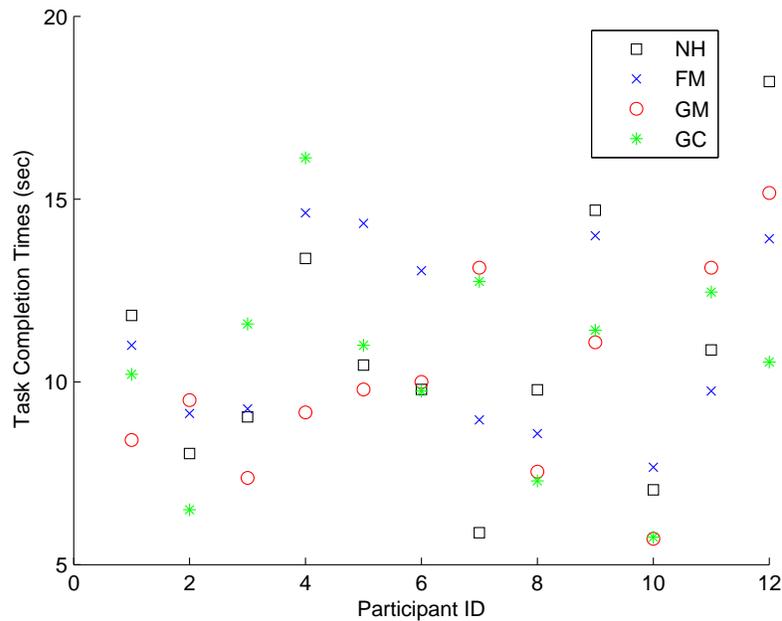


Figure 5.25: Average task completion times per participant and policy.

to be distracted from their tasks.

Analogous to the simulation experiments, we report on the percentage of automated suggestions and the percentage of user acceptances in Table 5.18, presented in the format of “Percent User Acceptances/Percent Suggestions”. For more detail, we also include the same comparison broken down by individual participants in Figure 5.26.

Overall, we see that participants did not accept help from FM very often, and accepted about 40% of the suggestions from GM and GC policies. We suspect this difference is due to the sequential nature of the highlighting task. Recall that FM makes suggestions

Table 5.18: Average percentage of acceptances and suggestions per trial.

FM	GM	GC
11.7/100.0	41.2/62.1	38.6/47.4

based on the frequency of completed user goals. In other words, as the user executes various events within an episode, the system’s frequency distribution of completed goals is the same. In effect, this means that the suggestions made by FM will not change after each user event, even if the executed events in the current episode indicate inconsistencies with the suggestions in the toolbar. In contrast, GM and GC can change their suggestions *incrementally* based on each observed event. Indeed, several participants asked the experimenter why the suggestions in FM did not update the icons based on the executed events. Although the usefulness of FM as an adaptive policy for menu selection tasks has been studied [Tms05], we believe its utility is limited in sequential tasks due to its inability to update suggestions during task execution.

To gain a deeper understanding about the type of suggestions accepted by the participants, we analyzed the accepted suggestions and compared them to the target goals. In most cases, we found that participants accepted suggestions that were identical or nearly identical to the target goals. In particular, by counting the percentage of events that are different between the accepted macro and the target goal, we found that the percentages vary slightly under different policy conditions: FM differs by 5.9%, GM by 3.3%, and GC by 2.5%. Individual differences exist as we see in Figure 5.27. For example, participants 2, 7, and 12 accepted suggestions that required between 15% to 34% of the accepted events to be manually corrected.

After accepting partially correct help, participants are required to correct the highlighting pattern to match that of the target goal. Figure 5.28 illustrates the average number of events in this correction phase by policy and participants. On average, the

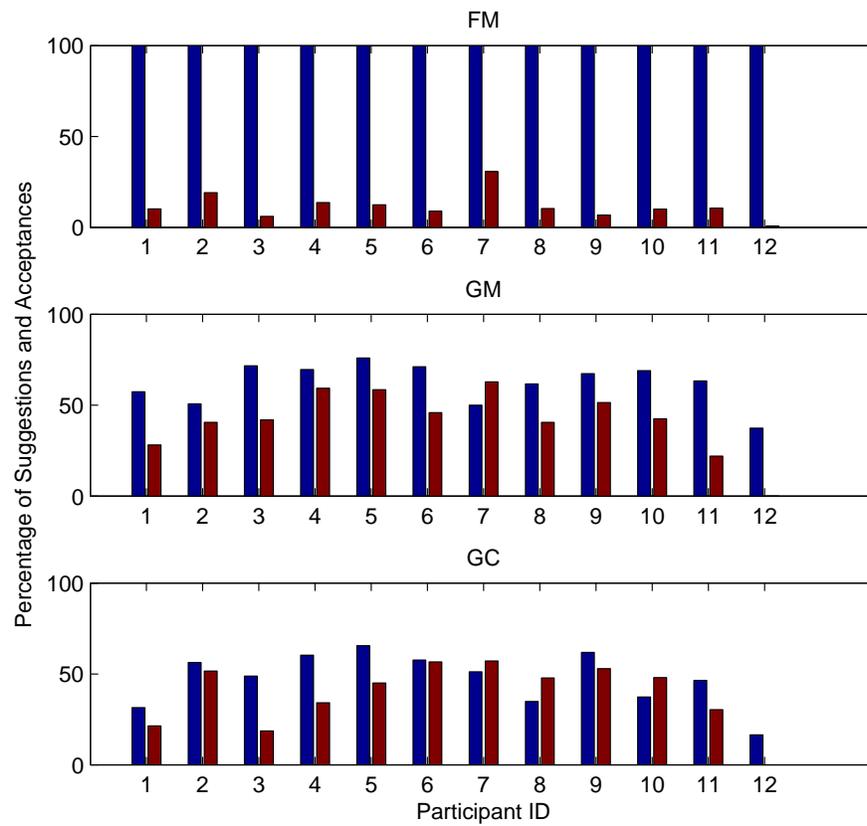


Figure 5.26: Percentage of suggestions and acceptances by participant and policy.

number of events corrected in FM is 1.16, in GM is 0.65, and in GC is 0.58. This pattern can be attributed to the results discussed earlier where participants accepted suggestions that varied in accuracy across the different policy conditions.

At the end of each session, we asked participants to self-report their perceptions of the system based on their experience. In comparing the four systems, we asked participants to indicate on a scale of 1 to 5 how frustrating it was to use the system, how easy the system was to use, and to what extent they liked using the system overall. Figure 5.29 shows that on average, GM is rated similarly to GC, and both are less frustrating, easier to use (or about the same), and better liked than NH and FM. In particular, participants consistently rated GM better than FM on these measures. However, we

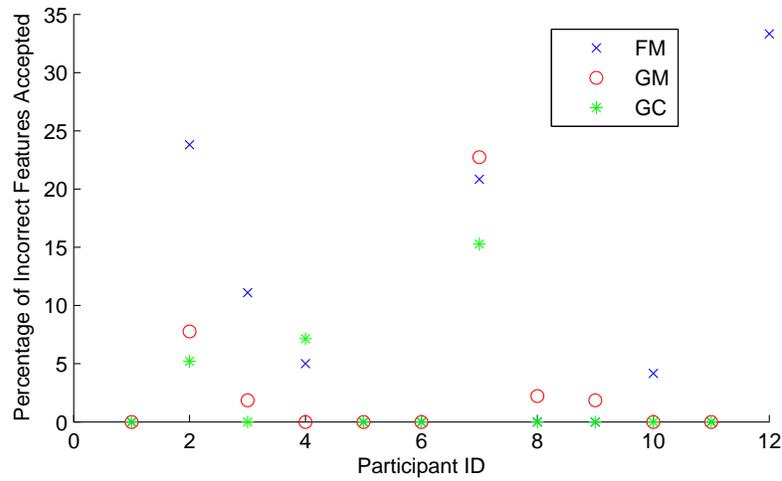


Figure 5.27: Percentage of incorrect features in automated suggestions accepted by participant and policy. Note that participant 12 did not accept help for the GM and GC policies, so those data points are not present.

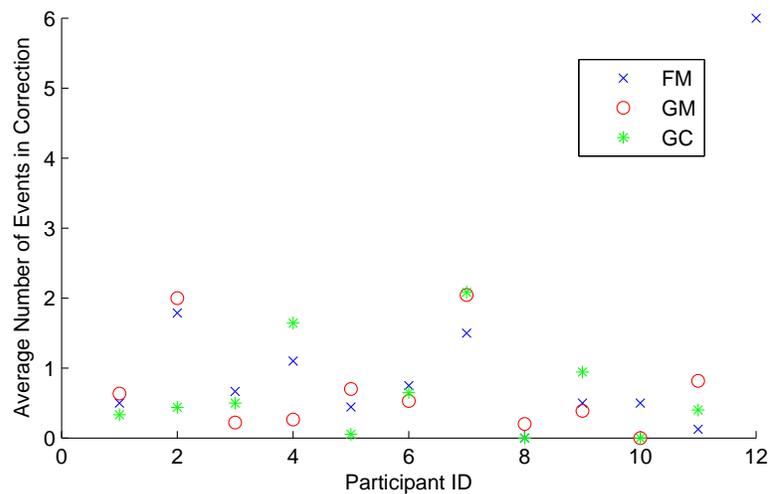


Figure 5.28: Average number of correction events executed by participant and policy. Note that participant 12 did not accept help for the GM and GC policies, so those data points are not present.

found no difference between GM and GC.

Lastly, the questionnaire also asked participants to indicate their perceptions of the

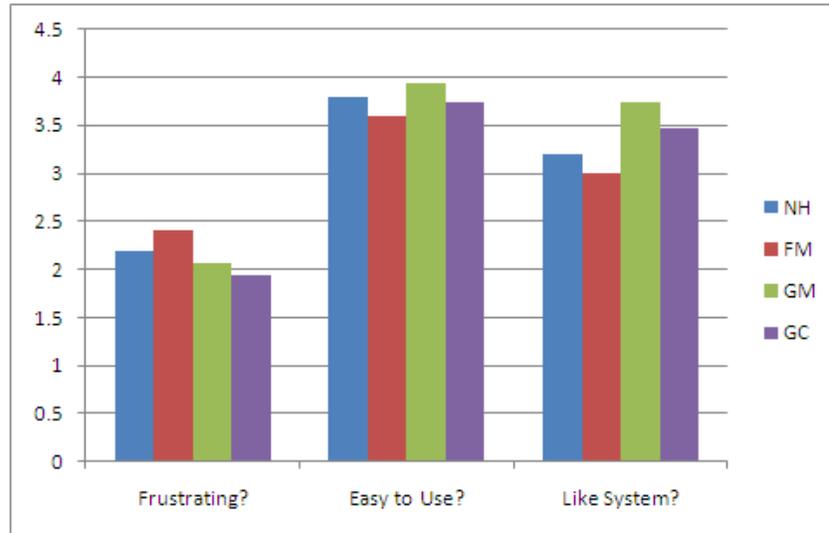


Figure 5.29: Comparison of all four policies.

automated suggestions in the three conditions with adaptive policies. The same response scale was used to ask how personalized the suggestions were to their preferences and needs, how helpful were the suggestions, how easy was it to determine whether icons in the suggestions were useful, and to what extent they agree that not having any suggestions is better. Figure 5.30 shows that on average. For these measures, participants rated GM and GC similarly, and rated both of them better than FM in all four aspects. In particular, GM is almost a full rating scale point better than FM in terms of personalization and helpfulness, but there was no difference between GM and GC.

5.6 Summary

In this chapter, we described an incremental goal recognition approach that personalizes to specific users' goal execution patterns in a realistic application. In the context of PowerPoint, we identified a set of goal classes that an intelligent system could be designed to help the user with. Among these, we focused on a highlighting goal class, and developed the recognition module for it. The module uses deterministic finite state automata to

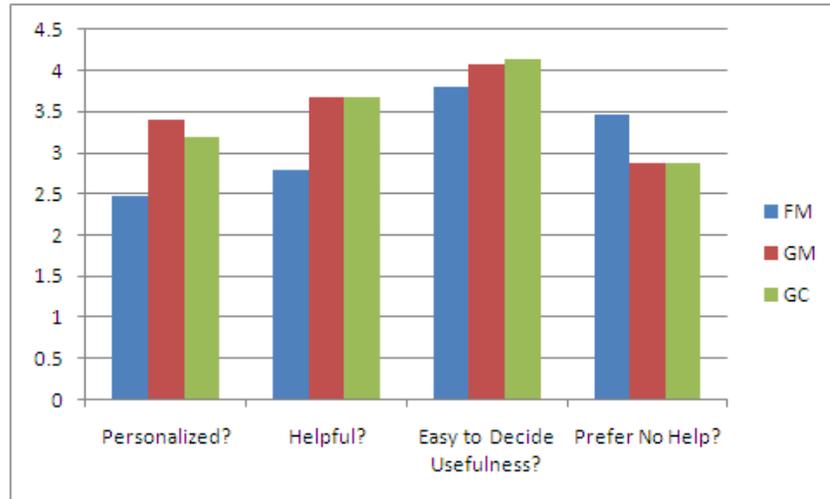


Figure 5.30: Comparison of three adaptive policies.

recognize highlighting goals and a probabilistic component to update the system’s belief of the user goal using the observed interaction history. In Section 5.4, we designed alternative decision policies that did not make use of distributional goal information and/or distributional user characteristics information. These policies were embedded in an adaptive version of PowerPoint that offers highlighting suggestions in a pop-up toolbar. Through simulation experiments, we found that policies that incorporated user characteristics information adapted to different user types better than alternative policies. Also, we found that policies that incorporated distributional goal information had higher expected utility than alternative policies. Our preliminary user study also supports these results. In particular, we reported in Section 5.5 that users generally accepted more help from our adaptive policies than an alternative policy that did not use a goal model. Moreover, the accuracy of the accepted help is better in our adaptive policies, resulting in fewer event corrections on average. These empirical results are encouraging, despite the system employing a slow inference engine. We suspect that a central factor that contributes to the success of our adaptive policies in this setting is due to the sequential nature of the highlighting task. In particular, since our adaptive policies are able to tailor

its suggestions upon every observed event, suggestions made before a goal is completed may improve as the user executes more events. On the other hand, the alternative policy (based on the frequency of *completed* goals) is unable to adapt its suggestions during goal execution. Thus, several users “complained” about that policy not being “smart”. This aspect of goal assistance introduced an unexpected variable into the study, so we are currently unable to attribute the success of the goal policies to their decision-theoretic formulation. This issue deserves further investigation and development. Overall, we demonstrated the development and evaluation of a general goal recognition model in the context of the User Goal Component within our DAISI framework.

Chapter 6

Utility of Intelligent Assistance

This chapter focuses on the reward model that plays the key role in the Action Selection Component in the DAISI framework as described in Chapter 3. As mentioned in Chapter 2, existing intelligent systems that adopt a DT approach model benefits and costs of system actions typically in a myopic fashion. When interaction costs are modeled, they are generally grouped into an umbrella term labeled as “cognitive cost” without consideration of the various dimensions that may be involved. One exception is the work by Gajos et al. [GCTW06] in which they studied the utility of several interaction dimensions as perceived by the user to explain interface preferences for several versions of split menus. Examples of interaction dimensions used include perceived mental demand, perceived frustration level, and perceived confusion due to adaptation. Although these concepts may inform designers as to how users perceive an interface after some experience with it, without definitions and knowledge about possible dependencies among these concepts, it is difficult to formalize them in an intelligent system that computes the expected utility of actions. In contrast, our approach focuses on formalizing different interaction factors drawn from pre-established interaction design criteria in the HCI literature. Where possible, we adopt existing predictive models for these factors. When such models are not available, we conduct empirical experiments to derive a model. Ad-

ditionally, we incorporate a subjective component in our predictive models based on the user characteristics discussed in Chapter 4 in order to account for the user’s perception of the various interaction factors.

Another distinction from the work of Gajos et al. [GCTW06] is the way their utility dimensions are combined. In their work, subutilities from each dimension are defined in the same range and then an equally weighted average of the terms is used as the overall utility of an interface. This implies that everyone perceives mental demand just as importantly as frustration, confusion, etc. However, such a restriction is not necessary; one user may value perfect help (i.e., quality of savings) much more than the number of help suggestions made (i.e., frequency of suggestions), while another user may find long lists of suggestions highly cognitively demanding to the point that the availability of perfect help is simply not worthwhile. A simple solution is to take a weighted average of the utility of each interaction factor, allowing weights to express individual preferences.

For these reasons, we propose to formally model the cognitive cost induced by a system action as different interaction factors in a DT setting. First, we discuss the set of intelligent system actions of interest and identify a set of relevant interaction factors from the HCI literature in Section 6.1. As a result, the factors we focus on are: interruption, information processing, target selection, visual occlusion, interface bloat, disruption, and savings. Given these interaction factors, we present the utility structure used in our DAISI reward model in Section 6.2. Since different users may perceive the impact (i.e., costs or benefits) of system actions differently, we explain how user characteristics variables allow us to model the subjective utility of an interaction factor, and their role in the Action Selection Component. For each interaction factor, we define a corresponding utility model using an influence diagram that lets us incorporate a predictable, objective value of that factor, as well as its perceived, subjective utility. In Section 6.3 to Section 6.6, we formalize each of the identified interaction factors and adopt existing predictive models where available. A summary of the model used and the experiments

conducted related to the benefits of savings from Section 4.3 is also included as part of this chapter. Among the set of factors of interest, we found that visual occlusion, bloat, and disruption to the user's mental model are often mentioned in the literature. However, to our knowledge, no existing research has developed predictive models for these three factors. As presented in Section 4.4, we explored the concept of disruption in depth and conducted various empirical experiments to learn model parameters. In addition to developing an objective model for disruption, these experiments also gathered data regarding a user's perceived cost of disruption in the context of adaptive menus. Since the details are documented in a previous chapter, we include a summary of the experiment and results in this chapter only. In an effort to develop formal models for various interaction factors, we conducted two empirical experiments to learn these predictive models and demonstrate one of them in a simulation experiment. These experiments are presented in Section 6.5 and Section 6.6.

The most common method used to collect user feedback on the subjective utility of an interface is the use of post-questionnaires (e.g., [GCTW06]). Although such questionnaires allow designers to gain a better understanding of the system's performance with respect to specific variables in question, they do not indicate how user preferences change as a function of the changes in these variables, in isolation or in combination. For this reason, we develop an experiential preference elicitation procedure for interface customization in Section 6.7. Following the work in Chapter 5, we use the example of suggesting highlighting help in a pop-up toolbar in the context of authoring slides using PowerPoint 2003. Background on the theory of preference elicitation, the details of the experiment, as well as the results in how different users have varying preferences in this domain are reported in Section 6.7.

6.1 Identifying Interaction Factors

In order to develop the relevant dimensions of the DAISI reward model, we need to identify the types of intelligent system actions of interest and the relevant factors of interaction that are related to these actions. In contrast to static interfaces, intelligent systems differ in that the interface may change as a result of specific system actions. For example, at a certain point in time, the system may add an icon to an existing toolbar on the interface, or decide to suggest auto-completion help while the user is typing a particular phrase, or both. An intelligent system must be able to evaluate the amount of effort its actions could save or cost the user [Hor99a], the amount of bloat its actions induce on the interface, as well as other kinds of interaction benefits and costs. To identify such benefits and costs, we consider interaction factors related to specific types of system actions and model the impact of each action explicitly. In this way, when the overall impact of an action is taken into consideration, the system has the flexibility to decide which action or combination of actions yield the highest expected utility. Note that a static interface is simply a special case of an adaptive interface with no action. Thus, the DAISI framework will generalize to static systems as well.

As mentioned in Section 1.1, the general objectives of an intelligent software adaptation system is to design software that helps users by minimizing the effort expended in carrying out tasks and/or maximizing the ease of interaction while using the system. We define several types of system actions that help achieve these objectives and consider their impact on the user below. In the case where a system is designed to help minimize effort, the system may take actions such as auto-completing the goal for the user (we refer to this type of action as *AUTO* for short), suggesting several possible completions as icons in a toolbar near the user's cursor to allow for convenient selection (*TBAR*), adding a permanent icon on the interface to allow the user to select it at anytime (*ADD*), and providing a hint that an existing function can do the desired actions in fewer steps (*HINT*). On the other hand, to maximize the ease of interaction, the system may take ac-

tions that help rearrange parts of the interface, such as moving frequently used functions (i.e., icons and menu items) to a more convenient area (MOVE), and hiding functions that are rarely used (HIDE). Moreover, to gather explicit information about the user's intentions or preferences, the system can ask a query (ASK). Note that these kinds of actions are not new; as examples, Microsoft Office tools have adaptive variants of AUTO, TBAR, HINT, HIDE, ASK and a user-defined version of ADD [Mic]. In addition, research on split menus have proposed variants of MOVE [SS94].

For each type of system action, we identify the relevant interaction factors that influence how users perceive those actions in an intelligent system. Based on previous literature on interface design principles and principles of interaction with adaptive assistance systems, the following is a preliminary list of the most relevant interaction factors using the terminology from the literature: bloat (i.e., unwanted functionality), dismay, satisfaction, disruption to the user's mental model, processing (i.e., perception, cognitive processing, decision time, eye movement, search time), selection (i.e., pointing movement), error rate, execution time, ease of use, discoverability, mental demand, physical demand, performance, efficiency, frustration, confusion, control, predictability, expectations, trust, safety, privacy, interruption (e.g., based on task concentration), visual occlusion, reassurance value, effort savings, and time savings [BGBG95, Nor94, McG00, SS94, HA03, War01, GCTW06, CCH01, AB04, SM97, CGG07].

In most cases, these concepts are discussed using intuitive scenarios and examples. Since our goal is to formalize interaction factors for the purpose of designing an intelligent software adaptation assistant, we merge concepts that seem similar or redundant (e.g., dismay may be treated as the opposite of satisfaction; reassurance, predictability, and expectations all address the same underlying concern) and eliminate ones that are not well-defined (e.g., perceived confusion) or not well-suited to the customization domain (e.g., safety, privacy). As a result, we remove the following factors from our discussion: dismay, mental demand, frustration, confusion, discoverability, reassurance value, trust,

safety, and privacy. Although we eliminate frustration, in fact, we can model frustration as a user characteristics variable and probabilistically infer the user's current level of frustration and use that information in the system's reasoning. Also, the notions of time savings, effort savings, physical demand, efficiency, and performance, as well as mental demand and disruption, have been merged. To keep our attention focused on interaction principles, we eliminate trust as a factor, although it could arguably be used as an important design criterion for software adaptation agents as well. This leaves the following interaction factors as central to our investigation: bloat, processing, selection, interruption, perceived savings, visual occlusion, and disruption.

Given the types of system actions and the interaction factors identified, we discuss the impact that each type of action has on the user, with respect to each of these factors. The seven types of help actions have different costs and benefits. Since these system actions are designed to help the user in some way, each action will have immediate and/or long term savings. By offering help, the system has opportunities to ease the interaction process and save the user the physical effort of manually carrying out each step of the task each time the task needs to be done. On the other hand, in order to let the user know that help is provided by the system, these actions also interrupt the user's current work flow when they are suggested. The specific cost of interruption varies with the implementation of that system action. In cases where a function is added, moved, or hidden, the user may not be interrupted unless a notification is implemented to let the user know of that change. For example, adding a new menu item may go unnoticed because menus are generally hidden until they are selected. However, adding a new icon may interrupt the user immediately if the toolbar is always present on the screen. Thus, interruption may be present for each of the action types, but can be eliminated for ADD, MOVE, and HIDE.

Although suggesting a toolbar near the user's cursor makes it easier to select, the toolbar itself will inevitably occlude a part of the content in the user's task. The same is

true for HINT and ASK actions. In some implementations of AUTO, where a completion phrase is suggested above the current user focus and requires the user to select the completion, occlusion is also relevant. As well, actions that present options for the user to evaluate and select have impact on processing and selection costs. For example, the more options that are available in a suggestion toolbar or a question, the more time it takes the user to process those options. Moreover, selecting an option that is placed further from the user's focus has a higher cost than if it were located closer. In general, HINT actions may simply include a text explanation to help the user, or they may include links to further details on that explanation. The latter case has an associated selection cost. Relevant to interface bloat are the addition and hiding of functions. Lastly, ADD, MOVE, and HIDE result in changing the location of functions. Therefore, they all cause spatial disruption to what the user already knows about those function locations. As an alternative, implementing notifications or animations can ease that disruption, but these methods will also have associated costs for processing and/or interruption.

Table 6.1 summarizes this discussion. This work, of course, provides merely a starting point in identifying interaction factors that form a basis for the user's reward function. As new intelligent system actions are introduced and additional interaction factors are deemed relevant, the impact that the actions have on users can be included in the same way.

Since much work has been focused on modeling interruption and related costs (such as attention, workload, workflow, concentration, urgency) [CCH01, AB04, HKA04, HA03, HKPH03, FHA⁺05], we do not discuss it further in this chapter. The next step in developing our DAISI reward model is to formalize these interaction factors and incorporate into the model a method to explain individual differences among users.

Table 6.1: Relevance of interaction factors for each type of help action. Notation: “x” indicates relevance, “(x)” indicates optional relevance depending on the specific system implementation.

	Processing	Selection	Occlusion	Bloat	Savings	Disruption	Interruption
AUTO		(x)	(x)		x		x
TBAR	x	x	x		x		x
ADD				x	x	x	(x)
HIDE				x	x	x	(x)
MOVE					x	x	(x)
HINT	x	(x)	x		x		x
ASK	x	x	x		x		x

6.2 Utility Structure

The DAISI reward model represents the user’s utility function in the intelligent software adaptation problem. For simplicity, we assume the reward model has a generalized additive structure, where the utility of each relevant interaction factor is represented using a *local value function*. To express the subjective aspect of the user’s utility function, each local value function is parameterized by a set of user characteristics variables. For example, the value function for occlusion is parameterized by the user’s level of frustration and distractibility. Thus, although a system pop-up action induces some amount of occlusion, the subjective cost of that occlusion depends on two user characteristics. In this way, the value of a system action pertaining to each interaction factor can be estimated *objectively* (say, using a predictive model learned empirically), and then modified based on the user characteristics parameters relevant to that interaction factor to express *subjective* utility using the user characteristics as the current context.

In cases such as information processing, selection, and savings where empirical models are available, we discuss how they are used in our model in Section 6.3. We adopt these models for estimating the objective component of the interaction factor. In other cases such as disruption, occlusion, and bloat, where the literature does not provide a predictive model for our purposes, we conduct preliminary experiments that learn a quantitative model for these interaction factors. These interaction factors and corresponding experiments are detailed in Section 6.4, Section 6.5, and Section 6.6 respectively.

Based on the work presented in Chapter 4, the reward model we develop here will make use of user characteristics variables such as the user's level of frustration, neediness, distractibility, independence, and strength of the mental model state. Recall that some of these characteristics have been proposed by other researchers as well (e.g., neediness and distractibility by [HBH⁺98], frustration or distress by [KBP07, CM05], and mental model state by [Sas97]). As we consider additional interaction factors in this chapter, we will also incorporate user characteristics variables such as the user's tolerance to bloat to express the subjective utility of interface bloat [McG00] and the user's expertise in information processing to express the subjective utility of processing time [CGG07].

By modeling the impact of each interaction factor using separate value functions, we are assuming that these factors are independent of each other. As an example, consider the cost of processing and the cost of interruption due to a system action that suggests a toolbar of icons on the screen. In this case, processing time refers to the duration of time required for the user to scan the items in the toolbar and either identify one to select or decide that all the items should be rejected. On the other hand, interruption refers to the instantaneous cost associated with the toolbar popping up on the screen, regardless of whether the user processes the items in the toolbar. As a result, we conjecture that independence holds (at least as a first approximation) for the remaining components. However, if future work indicates that certain combinations of factors demonstrate a compound effect, we can model those factors jointly using a single value function and

still assume additive independence of the remaining factors. In this work, we assume an additive structure for the DAISI reward model. Also, since some factors (such as frustration, distractibility, and concentration) are shared across the value functions, the reward model uses a *generalized additive decomposition* [Fis82, BG95].

The overall DAISI reward model is illustrated in Figure 6.1, with each value function expressed as an influence diagram. These diagrams show the variables modeled and the structural relationships among them. At a high level, the leftmost component modeling processing time in Figure 6.1 shows *Length* — the number of items to be processed — influences the variable *Processing*. This variable corresponds to the objective time required for the user to process information. Mathematically, we define *Processing* as a function of the input parameter *Length*. In turn, the objective variable *Processing* and the user-dependent variable *Speed* influence the subjective value of processing V_P illustrated in the diagram. We review the details of the value function for information processing in Section 6.3. The remaining components illustrated in Figure 6.1 are interpreted in a similar manner and are also discussed in more detail in subsequent sections.

In combining these components, the resulting subjective values of each interaction factor need to be calibrated against each other to reflect their relative importance based on the user’s attitude toward the respective interaction factor. For example, a user may not mind processing a lot of information (e.g., answering extensive questions) but is extremely bothered by spatial changes when interface widgets are moved to new locations. One way to model these individual differences is to define the local value functions in different ranges. In the previous example, the value for information processing may be defined in $[-5,0]$, while the value for disruption may be defined in $[-10,-5]$. Alternatively, local value functions may be weighted [KR76]. In the previous example, let v_p denote the value of processing and v_d denote the value of disruption, these two values may be defined in the same range and then combined into a single utility via $w_p v_p + w_d v_d$, where the individual weights w_p and w_d express the relative importance of the two interaction

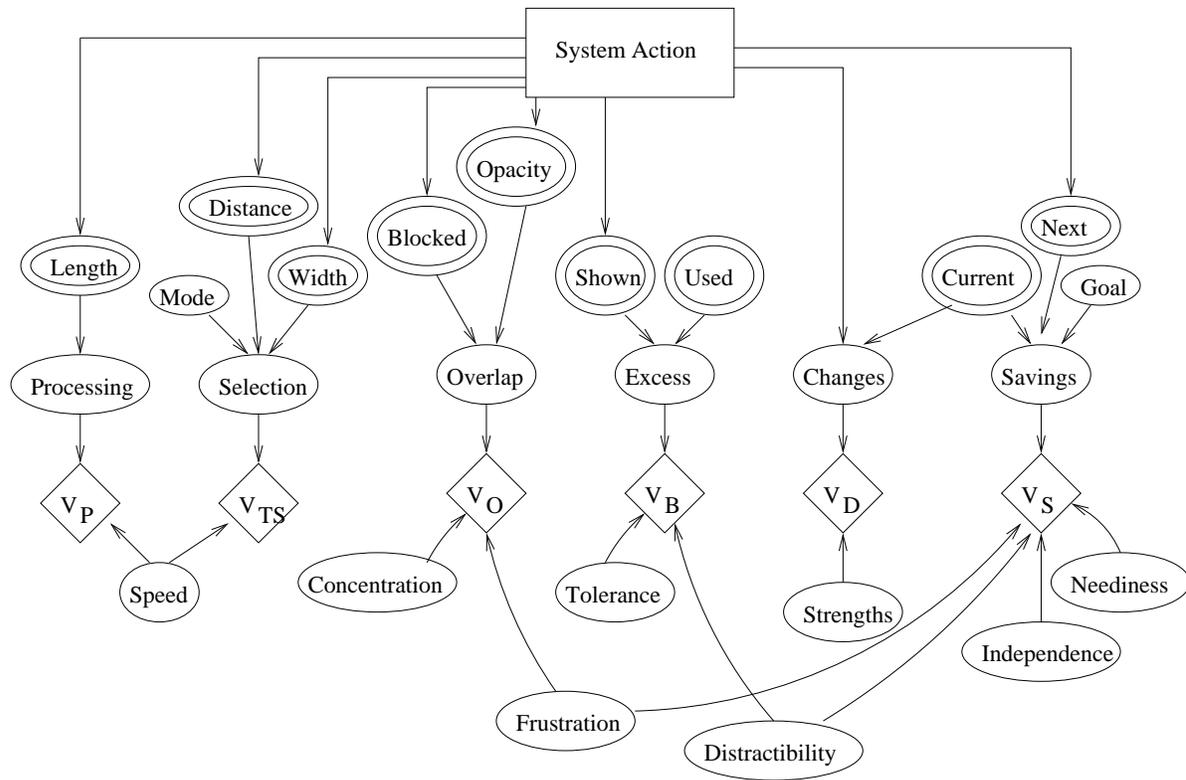


Figure 6.1: A graphical abstraction of a generalized additive reward model. The reward model consists of independent local value functions shown as influence diagrams. This model includes processing (V_P), target selection (V_{TS}), occlusion (V_O), bloat (V_B), disruption (V_D), and savings (V_S), and excludes the cost of interruption. Utility nodes are denoted as diamonds, unobserved variables as ovals, and observed variables as double-circled ovals.

factors for a specific-user.

Obtaining the subjective value of each local value function should ideally be validated through extensive preference elicitation experiments. One difficulty is the complexity of the problem because of the number of variables involved. Another difficulty lies in controlling for the unobserved user variables in our model, such as the user's level of concentration, frustration, neediness, etc. In Section 6.7 below, we explore a new preference elicitation method that controls for the user's level of neediness and the tradeoffs involved

in perceived processing time and savings in the context of automated suggestions.

6.3 Adopting Models for Processing, Selection, and Savings

In this section, we review the interaction factors that have existing objective models from the HCI literature and explain how these models fit into our DAISI framework.

6.3.1 The Cost of Information Processing

Information processing has been heavily studied in cognitive psychology. In the context of intelligent software adaptation, system actions — such as suggesting a toolbar, asking a question, or displaying a hint — involve different kinds of processing costs (cf. Table 6.1). In particular, we consider toolbars that have one or more icons to chose from, text questions that have possible responses to choose from, and hints that range over several sentences. The cost associated with information processing is defined as the time when the user starts to scan and consider the available information to the time when the user is finished with it (which often leads to selecting an option included in the information). Thus, in the case of a toolbar, the information that requires processing is a set of icons, while the information that needs to be processed is text for the other two system actions. As well, combinations of icons and text can be used in any of these system actions.

Existing literature suggests that the time required by experts to react to a set of possible choices can be modeled by the Hick-Hyman Law [Hic52, Hym53]. Intuitively, this model suggests that expert users know the interface well and can rely on memory from previous experiences to react to the options more quickly. However, in intelligent systems where system actions are adaptive, the set of choices presented to users, as well as the location of where the choices are displayed, may differ each time. Thus, a more appropriate model involves visual search [HK97b] where users are viewed as naive and

must consider each option in turn. In situations where the user may be modeled as being a mixture between a novice and an expert, the processing time has been suggested to be a weighted average as follows [CGG07]:

$$T = (1 - \epsilon)T_{vs} + \epsilon T_{hh} \quad (6.1)$$

where T_{vs} is the time predicted using the visual search model, T_{hh} is the time predicted using the Hick-Hyman Law, and $\epsilon \in [0, 1]$ is a value expressing the user's expertise level.

Considering the choices available in toolbar and question actions more deeply, we see that further complications arise. In the case of a suggestion action where one or more icons are displayed in a toolbar, the user needs to consider whether any of the icons matches his goal. Using the toolbar of highlighting icons from Chapter 5 as an example, the skills a user would need to consider whether to select an icon or not involve visual matching and imagery recognition. For example, the user will have a target highlighting goal in mind, and when a set of icons are suggested, he must visually compare each icon until a match is found. However, this is a simple example involving only highlighting icons — combinations of font attributes — in the context of highlighting goals. In a broader context, the system could offer toolbar suggestions where icons could represent the execution of other goals and even other software applications. For example, the system might believe that the user is stuck while writing a technical definition, and decide to provide to the user an icon representing the web browser that opens up an online dictionary. (Similarly, asking a question where the user needs to evaluate multiple responses and select one can lead to complicated scenarios.) For simplicity, we model the evaluation aspect of information processing using a general mental cost constant. An example is the mental preparation cost suggested in the GOMS model [CMN80] (about 1.35 seconds for average users, see Section 6.3.2 for discussion). In this way, each option that is available for consideration has an associated mental cost. Thus, in accordance with the visual search model, the cost of processing is linear in the number of options present in a system action.

When a hint is displayed, it could show a text explanation and potentially a link to more details in a separate help component. In this case, the user could ignore the hint, or read the text in it and possibly select the link. Similarly, when the system asks questions, the question may consist of several sentences and a set of multiple choice responses for the user to choose from. For reading times, simple models have shown that reading speed is linear in the number of words, and that the average reading speed is about 3.75-4.5 words per second (or, 0.27-0.22 seconds per word) on a computer screen (10% slower than reading on paper) [Bai96]. Of course, if the sentences have complicated structure or either syntactic or semantic errors, then reading time is increased. If a link is present, then the target selection time needs to be modeled.

Going back to influence diagram for processing in Figure 6.1, we see that *Length* is an input parameter for estimating the objective process cost, i.e., $Processing = f(Length)$, where f is a linear function, and *Length* is the number of items to be processed (possibly of different types). Since this is a general model that accounts for processing text and iconic information, the specific constants in the linear function need to be determined according to the particular context. Once the objective cost of processing is estimated, we can use the variable *Speed* to capture the individual differences in the subjective value of processing. This variable is akin to the expertise level used in [CGG07].

6.3.2 The Cost of Target Selection

Target selection has received much attention in HCI. System actions that often require the user to perform target selection include suggesting a toolbar, displaying a hint, or asking a question. In these cases, the target may be selected using the mouse or keyboard. We consider target selection as the effort in the physical action of selecting a target after knowing what the target is. For simplicity, we assume that target recognition (which involves information processing) and target selection are carried out sequentially, and focus strictly on target selection cost here.

Table 6.2: Elementary actions in GOMS with suggested cognitive processing times.

	Acronym	Action	Time (seconds)
1.	K	key press and release	0.28
2.	P	point and move cursor to a target location	1.10
3.	B	button press and release on mouse (or any device)	0.20
4.	H	hand preparation for mode switching	0.40
5.	M	mental preparation	1.35
6.	W(t)	wait time for system processing	t

The objective cost associated with target selection is measured using total time, defined as the time when the user begins to take actions (using the mouse or the keyboard) toward selecting the target until the target is selected. We first review a general model that considers different interaction modes and then turn to one that focuses on mouse interaction only.

GOMS is a cognitive model that describes the general human computer interaction using a set of basic user actions [CMN80]. One variation of this model is called KLM-GOMS, which models interaction at the keystroke level and uses these user actions to estimate average task performance time for an interface. KLM-GOMS has six elementary actions shown in Table 6.2. Average empirical performance times are associated with each of these actions and total task execution time is calculated as the sum of the times of each action involved. Different literature sources may provide associated times that vary slightly, as empirical results will vary with different user samples.

As an illustration, we show how KLM-GOMS decomposes the task of selecting text and the times associated in the task in Table 6.3.

Note that KLM-GOMS uses a general pointer action that does not account for the distance between the user's current cursor location and the target location. As an alternative, the suggested times for pointer actions can be replaced by estimated times

Table 6.3: An example of KLM-GOMS analysis for selecting text.

User Action	KLM-GOMS Action	Time (s)
move cursor to beginning	P	1.10
click mouse button	K	0.20
move cursor to end of phrase	P	1.10
verify selection	M	1.35
total time		3.75

Table 6.4: Empirically derived average constants for two-dimensional Fitts' Law [Mac95].

	<i>a</i>	<i>b</i>
Dragging model	135	249
Pointing model	230	166

according to Fitts' Law [Fit54], which models the user's rapid, aimed movement using the mouse. The two-dimensional formula is defined as:

$$MT = a + b \times \ln\left(\frac{D}{W} + 1\right) \quad (6.2)$$

where MT is the average movement time, a and b are empirical constants obtained via a regression fit, D is the distance from the starting point to the center of the target, and W is the width of the target. In intelligent systems, we want to estimate MT . Given the specific system action, the system can determine the values for D and W . For the empirical constants a and b , we can use the values proposed by [Mac95] shown in Table 6.4. These values can be adapted online based on observed interaction over time.

A general target selection model could use KLM-GOMS to model movement times for keyboard interaction and Fitts' Law for mouse interaction. In the influence diagram for selection in Figure 6.1, the objective selection cost is $Selection = f(Mode, Distance, Width)$, where $Mode$ is a binary indicator variable referring to the mode of interaction (mouse or

keyboard), f is either the function that sums up the relevant times according to KLM-GOMS or the logarithmic function according to Fitts' Law, and *Distance* and *Width* are the input parameters D and W used in Equation (6.2). Once the objective cost of target selection is estimated, we can use the variable *Speed* to define the individual differences in the subjective value of target selection.

6.3.3 The Benefits of Savings

We consider the physical effort saved in accepting automated help as the core benefit for all the intelligent system actions. Depending on the user goal, the type of physical effort savings may vary. For example, automated completion actions can save users from having to type additional characters in a word or a phrase, while accepting a highlighting macro from a toolbar suggestion can save users from having to use the mouse to select a set of font attributes individually to highlight a phrase. Both of these examples indicate immediate savings that result from an intelligent system action. On the other hand, the remaining types of system actions discussed in Section 6.1 have savings that get realized in the long term. In particular, adding a new menu item or icon to the interface, hiding an unused menu item or icon, moving a menu item to a more convenient location, hinting to the user that there is a faster way to do something, and asking the user a question to better tailor the system according to the elicited preferences all system actions that are intended to save the user effort in future interactions. To model long term savings, we can use a discount factor and an infinite horizon in expectation of a distribution over function usage. An example of this was demonstrated in Section 4.4 in the context of adaptive menus. Here, we focus on objective models for immediate savings.

Restricting our attention to mouse and keyboard input devices only, we define savings as the physical, manual effort required by the user to change the current application state to a desired goal state. Examples discussed in previous chapters include the effort exerted in typing the remaining characters of the intended word and the effort exerted in selecting

the remaining font icons of the intended highlighting goal.

Note that this definition of savings overlaps with the definition of target selection from Section 6.3.2 when the savings involved pertain to the task of selecting a target. Here, savings encompasses a broader definition because the user's goal may not relate to target selection at all (e.g., such as typing a word). While we include both of these interaction factors in our model, in practice, if the system's decisions pertain to target selection tasks, then the designer needs to ensure that the costs and benefits in those decisions are not counted twice.

Models for estimating the objective savings in terms of the time associated with the effort include GOMS and Fitts' Law, restricted to their applicable contexts as discussed in Section 6.3.2 above. Alternatively, rather than modeling task execution time, we could model the events the user need to execute to change the current application state to the desired goal state [FLL02a]. As mentioned in previous chapters, we demonstrated both approaches in different case studies. For more detail on how we model the value of objective savings, the reader is referred to Section 4.3 on modeling savings for typing tasks, Section 4.4 and Section 6.6.2 for menu selection tasks, and Section 5.4.1 for highlighting tasks.

The influence diagram for savings in Figure 6.1 shows that the objective savings is defined as $Savings = f(Goal, Current\ State, Next\ State)$, where *Goal* is the system's representation of the goal state, *Current State* is the representation of the current state, *Next State* is the representation of the next state based on the suggested system action, and f is a function that calculates the effort required to get from the current state to the goal state, according to the available user actions defined in the particular application. Once the objective savings is estimated, we can use variables such as *Frustration*, *Neediness*, *Distractibility*, and *Independence* to define the perceived savings of system actions. For more detail on how we modeled the perceived savings, the reader is referred to the case studies in the four sections mentioned above.

6.4 The Cost of Disruption

As discussed in detail in Section 4.4, disruption cost is a central problem in user acceptance with adaptive systems due to the immediate, as well as long term, effects adaptive actions have on the user’s mental model of the application. Briefly, the user’s mental model of the application is a representation of the knowledge that the user has about the application, and disruption is a negative effect resulting from changes in the interface or functionality that conflicts with the user’s mental model. Following the example and work developed in Section 4.4, we discuss disruption in the context of adaptive menus. With this focus, the system actions that induce disruption are adding, hiding, and moving menu items. In particular, if the user knows the location of certain menu items very well, then moving those to another location (in the same menu or a different one) would cause a lot of disruption to the user’s mental model. As well, hiding frequently used menu items will also have a large disruption cost because users would have a hard time finding them the next time they need to be used. Finally, adding new menu items changes the relative locations of the existing items. Thus, the locations of other items that share the same menu are indirectly changed as a result.

The model proposed in Section 4.4 suggests that the subjective disruption cost is a function of the strength of the user’s mental model of the item being changed, and the type of change applied to it. For example, the system action MOVE may move a menu item from its current location to the top of the menu, thus, shifting all the other menu items that were originally above it down one spot. Alternatively, the system may move a menu item up one spot, thus swapping it with the item that was above it and causing that item to be shifted down. From these examples, several factors that influence the perceived cost of disruption can be identified — the type of location change involved, the impact this change has on the neighbouring items, and the strengths of the mental models of all these menu items. The influence diagram in Figure 6.1 shows perceived disruption is defined as a function of *Mental Model Strengths* and *Changes*. In the case

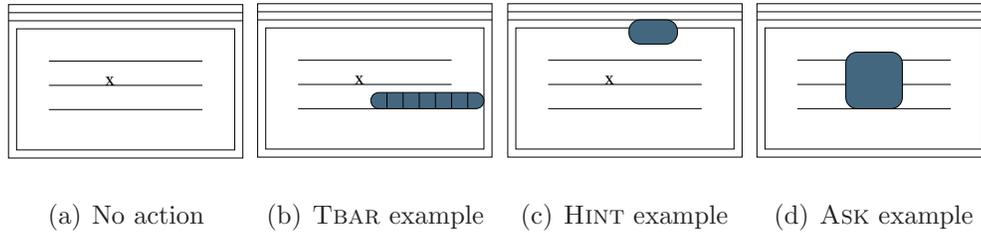


Figure 6.2: Different amount of visual occlusion created by three types of system actions. Notation: “x” indicates the user’s current focus.

of adaptive menus, *Changes* refers specifically to the relative changes in the locations of all the effected menu items, which result from the next menu state after applying the system action to the current menu state. For more detail on how the cost of disruption is formalized, the reader is referred to Section 4.4 for modeling disruption in the context of adaptive menu selection.

6.5 The Cost of Visual Occlusion

Visual occlusion is often mentioned in the design literature but, to our knowledge, there have been no formal attempts to model it. We define visual occlusion as the area in the user’s current workspace (such as the application window in focus) that is being blocked from view by an occluding object. As such, system actions such as toolbar suggestion, displaying a hint, and asking a question all induce visual occlusion. An example illustration of these situations is shown in Figure 6.2.

As a first attempt at modeling the objective cost of visual occlusion, we consider modeling the space, rather than the amount of information, that is occluded from the present view. In this way, the objective amount of occlusion is defined as the area occupied by the system action overlapping the user’s workspace. As an initial model, we consider the area in the application window of focus as the user’s workspace. The details of the experiment we conducted and the model derived is documented in Section 6.5.1. As a result, we obtain the model $Overlap = f(Opacity, Blocked)$, where *Overlap* refers to

the objective cost of visual occlusion, *Opacity* is the level of transparency of the occluding widget, *Blocked* is a binary indicator variable referring to whether the user’s immediate focus is occluded, and f is a linear function when $Blocked = true$ but a constant function when $Blocked = false$.

Given the objective occlusion cost, we propose that the subjective value of occlusion is influenced by the user’s level of frustration and level of concentration as illustrated in the influence diagram in Figure 6.1. For example, the larger the area of overlap, the larger the occlusion cost induced by the system action, and being highly concentrated and/or frustrated escalates that cost.

6.5.1 Experiments for Learning Occlusion

In this section, we describe the experiments conducted to empirically derive a quantitative model for occlusion [HGIB08]. The analysis of the experiment investigates the relevance of the tested independent variables and empirically derives a functional form to estimate occlusion. While occlusion has largely been neglected in experimental studies, our results show that it is an important factor in the user’s experience with an intelligent system.

In total, this experiment consists of 12 volunteer participants from a graduate computer science pool, all of whom have a good command of written English and no motor control deficiencies.

In this experiment, we simulate a typing task by asking participants to type specific letters in a sentence. In particular, each trial consisted of three parts. First, participants were asked to type the highlighted letter in the middle of the screen. We refer to the highlighted letter in the task as the *target*. Once that letter is pressed, a pop-up box (whose attributes are defined according to four test parameters below) and a new target appears. The participant may optionally close the box by pressing the “ESC” key. The trial ends when the participant types the second target letter. We measure the time between the two typed letters in each trial. A screenshot of this program is shown in

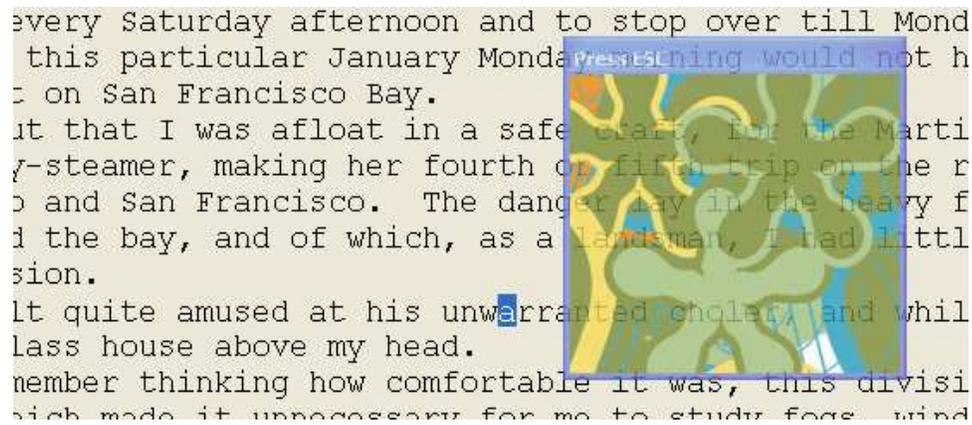


Figure 6.3: Screenshot showing a pop-up dialog box of size 200×200 pixels at 80% opacity in a typing task.

Figure 6.3.

The attributes of the occluding dialog box in each trial are defined as follows:

- *Dir*: The angle (direction) from the focal point to the center of the dialog box. Defined in 45° increments, this variable has eight values in total.
- *Size*: The width dimension of the dialog box. Starting with the size 50×50 , defined in 150 pixel increments, up to 500×500 pixels. This variable defines four unique size values in total.
- *Prox*: The distance (proximity) from the center of the dialog box to the participant's focal point. The values used are 100, 150, and 200 pixels.
- *Opacity*: The degree of transparency of the dialog box. This variable defines five values: 20%, 40%, 60%, 80%, and 100%, where 100% is fully opaque.

Altogether, this experiment has $|Dir| \times |Size| \times |Prox| \times |Opacity| = 8 \times 4 \times 3 \times 5 = 480$ configurations for the dialog box. With each of these configurations repeated three times and presented in a random order, participants were asked to complete a total of 1440 trials.

In addition to the above independent variable values, we recorded the overlapping space between the second target letter and the dialog box in two ways. First, considering only the target letter, we define *Lbox* as the intersecting area between the target letter and the dialog box. Second, considering the general area around the target letter to approximate the area of the user’s peripheral focus of attention, we define *Bbox* as the intersecting area between a 50×50 bounding box around the target letter and the dialog box. In summary, each trial provided a data point for task completion time, *time*, as a function of the following independent variables: *Dir*, *Size*, *Prox*, *Opacity*, *Lbox*, and *Bbox*.

To create a simpler model, we used factor analysis [Gor83] to obtain a subset of variables that approximate the data. Since *Bbox* and *Lbox* are not mutually exclusive, we converted *Lbox* into a binary variable called *Blocked* (to indicate whether the occluding box directly overlaps with the second target letter) and combined *Bbox* and *Lbox* into a tertiary-valued variable called *Blocked3*. Using factor selection guidelines (Kaiser criterion, the scree test, the percentage of variance explained, and the interpretability criterion), we compared different combinations of these independent variables. First, we found that *Blocked* provides the same amount or more information as any combination of *Blocked3*, *Bbox*, *Lbox*. Second, *Opacity* was the strongest indicator and *Dir* was the weakest. It is a bit surprising that *Prox* is not a strong indicator in this data. We suspect that this variable would be a stronger indicator if it had been designed to test for a wider range (e.g., 1000 pixels, 2000 pixels, etc.). In the end, our best candidates were the variables *Size*, *Prox*, *Opacity*, and *Blocked*, which explained 86% of the variance. In comparison, the variables *Opacity* and *Blocked* alone explained 84% of the variance. For simplicity, we picked the latter set of variables to form our model.

Using *Opacity* and *Blocked* as independent variables and *time* as dependent variable, we plot the data averaged over other variables and participants in Figure 6.4. The data corresponding to when *Blocked = true* and when *Blocked = false* is analyzed separately.

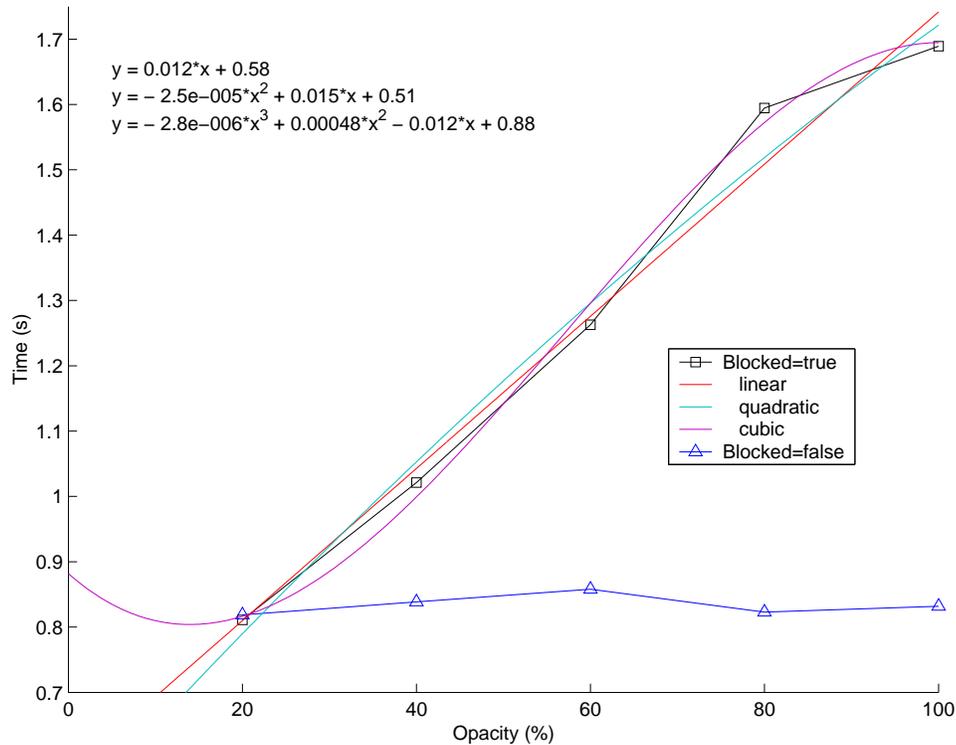


Figure 6.4: Observed and predicted average user performance for the occlusion task, showing linear, quadratic, and cubic regressions when $Blocked = true$.

Not surprisingly, when $Blocked = false$, we see from Figure 6.4 that there is almost no effect on task completion time. Thus, the data illustrates a (near) constant function in the figure. As an aside, a linear function fit to this data was:

$$time = .00005 \times Opacity + 0.83 \quad (6.3)$$

which is approximately $time = 0.83$. When $Blocked = true$, we see that higher levels of opacity increases task completion time. We conducted linear, quadratic, and cubic regression on the data in this case, and the resulting functions are graphed in Figure 6.4. By inspection, all three regressions provide a close fit. For simplicity, we adopt a linear model. The resulting function in this case is $time = .012 \times Opacity + 0.58$ with $r^2 = 0.981$.

6.6 The Cost of Interface Bloat

“In the popular press *bloat* is often used as a catch-all term suggesting that the software is filled with unnecessary functionality” [McG00]. Although this description suggests that bloat is the amount of currently unneeded functionality, it does not provide a precise definition. Here, we define bloat as the difference in the set of displayed functions accessible to the user and the functions that are actually used. As indicated in Table 6.1, the types of system actions that influence bloat are adding functions to the interface and hiding them. The change in bloat that results from these system actions has both an immediate and a long term effect. Just as we did in the case with modeling long term savings in Section 6.3.3, the use of a discount factor and an infinite horizon can model these long term effects.

As a first attempt to formalizing the immediate cost of bloat, we conduct an experiment to empirically derive a predictive model. In summary, the objective cost of bloat is defined as a function of the number of functions used in comparison to the number of functions shown, i.e., $Excess = f(Used, Shown) = Shown - Used$, where *Used* and *Shown* are the number functions used and shown respectively. We refer the reader to Section 6.6.1 for a justification of this definition.

It has been suggested that there are three levels of tolerance to bloat [McG02]. In particular, some users have a high tolerance and prefer to see all the functionality displayed even if it is not used. On the other hand, some users prefer a subset of personalized functions displayed, regardless of how often those functions are used. Finally, some users have a low tolerance and prefer a minimal interface consisting of functions that are used often. A simpler model suggests that there are two types of users instead: those who are feature-keen — corresponding to a high tolerance level, and those who are feature-shy — corresponding to a low tolerance level [MBB02]. We introduce a user characteristics variable *Tolerance* to model the user’s tolerance toward bloat. This variable can be defined as a tertiary or binary variable depending on the definition adopted. Thus, each

of these user types will “perceive” a different amount of bloat. Note that this level of gradation is simply a coarse approximation to how the variable should be defined (e.g., it may be defined as a continuous variable instead). In addition, an interface with many excessive functions may be distracting to some users. Thus, our subjective model also considers the user’s level of distractibility. Altogether, the subjective value of bloat is defined as a function of *Excess*, *Tolerance*, and *Distractibility*. Figure 6.1 illustrates this relationship, and Section 6.6.1 provides an example model and simulation results demonstrating the tradeoffs between bloat and savings.

6.6.1 Experiment for Learning Bloat

In this section, we describe the experiments conducted to empirically derive a quantitative model for bloat [HGIB08]. The approach taken in this experiment follows that of the one for occlusion as described in Section 6.5.1. In total, this experiment consists of 12 volunteer participants from a graduate computer science pool, all of whom have a good command of written English and no motor control deficiencies.

In this experiment, we designed a menu selection task with an interface that has the same menu structure as Microsoft Word with a total of 152 menu items. To separate our interface from possible experiences that participants might have with Word, our menus use abstract labels. In total, this experiment has four conditions defined by the following variable:

- *Shown*: The number of menu items shown. Defined values are 18, 62, 107, and 152.

In all four of these conditions, we fixed the number of menu items to be used by participants (*Used*) to 15, which is approximately 10% of the total (152). The target items in the selection task are randomized across conditions.

In each trial, participants follow an instruction (e.g., Fruits \rightarrow Papaya) and select the

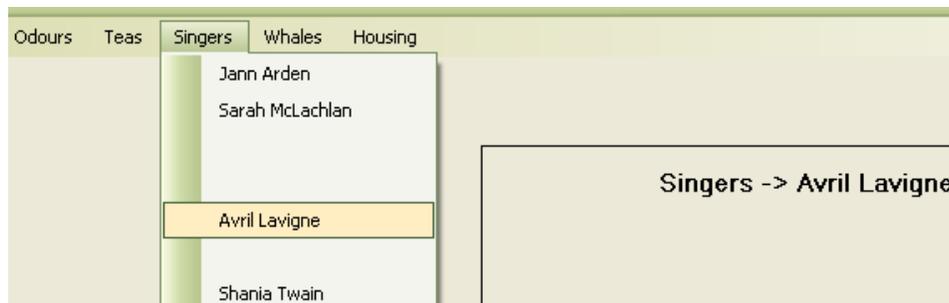


Figure 6.5: Screenshot showing the target menu item and instructions on the right. Notice this menu has many empty slots.

target menu item. Once the target is successfully selected, the participant clicks “OK” and the selection time is recorded. We measured the successful selection time of target menu items. A screenshot of the program is shown in Figure 6.5.

The experiment uses *Shown* as an independent variable and *Used* as a constant. Each condition required the participant to carry out 15 menu selection tasks, giving a total of 60 trials per participant. We counterbalanced the order of blocks using a size 4 Latin square. In summary, each trial provided a data point for menu selection time as a function of *Shown* and *Used*.

Note that *Shown* and *Used* can be combined in different ways to define the amount of bloat in an interface. As a first investigation, we were interested in the relationship between *Shown* and *Used*, and how this relationship can be used to define excessive functionality, *Excess*. We propose three candidate definitions for *Excess*: $Used/Shown$, $(Shown - Used)/Shown$, and $Shown - Used$. Using the data collected from the experiment, we plotted the task completion time as a function of *Excess* according to these three definitions shown in Figure 6.6(a-c) respectively. These graphs are averaged over participants and trials.

From Figure 6.6(a), we see that the definition of excess as $Used/Shown$ predicts that task times decreases as a function of increasing excess. Since this pattern contradicts

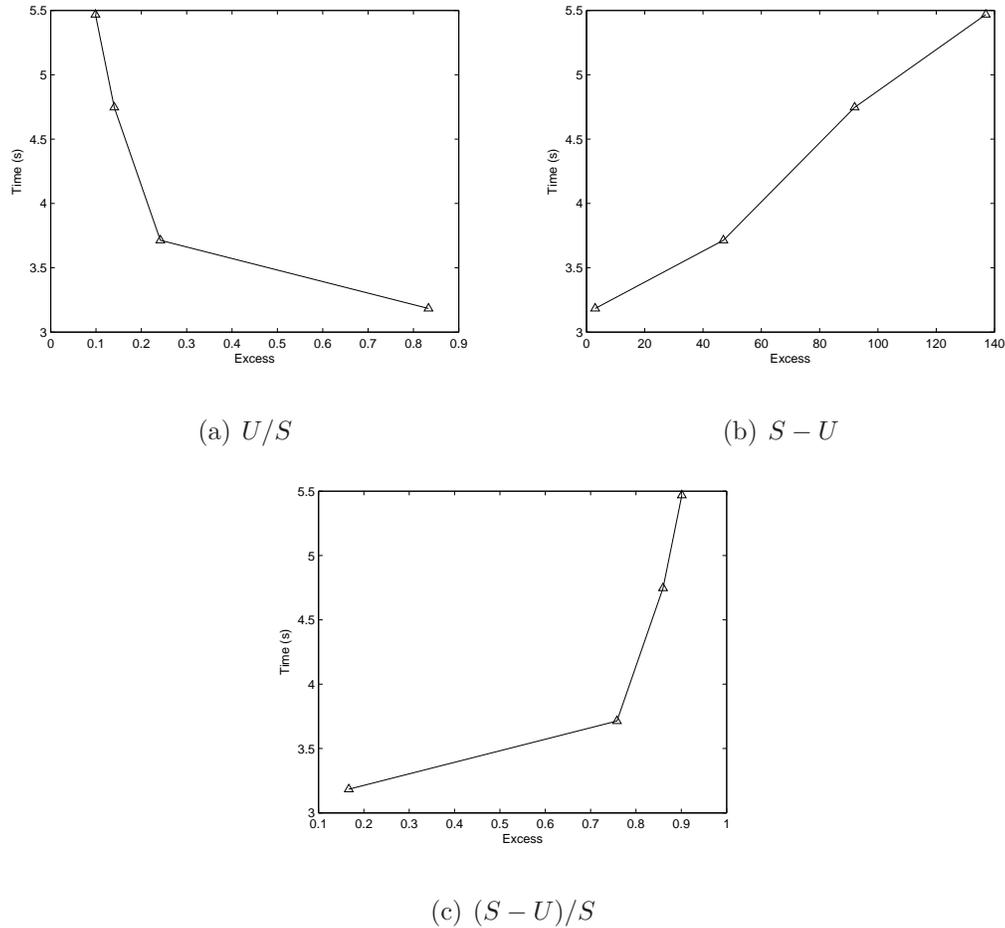


Figure 6.6: Menu selection times as a function of different definitions of *Excess*. Notation: *S* is short for *Shown* and *U* is short for *Used*. The x-axis plots increasing *Excess* to the right, and the y-axis plots increasing time upwards.

literature findings, we eliminate this formula as a viable definition of excess. Next, we compare the definition *Shown - Used*, where excessive functionality is defined as an absolute difference (shown in Figure 6.6(b)), to the definition $(Shown - Used)/Shown$, where excessive functionality is defined as a percentage (shown in Figure 6.6(c)). The main difference between these two definitions is illustrated by the examples in Table 6.5. While these two examples give the same percentage of excessive functionality, they result in very different absolute differences that reflect the amount of surrounding visual clutter in the interface. Although we did not control for these conditions in the experiment,

$Used$	$Shown$	$Excess = Shown - Used$	$Excess = (Shown - Used)/Shown$
10	100	90	.9
1	10	9	.9

Table 6.5: Examples that distinguishes $Excess$ defined as the difference versus the percentage of two variables.

we suspect that task completion times would have been impacted by clutter — thus, suggesting that the definition of excess as an absolute difference would better explain the data. Here, we adopt the definition $Excess = Shown - Used$, however, further studies are needed to confirm this choice.

By inspection, the graph in Figure 6.6(b) suggests a linear function model. With this data, we carried out linear and quadratic regressions on the data averaged over participants and trials as shown in Figure 6.7. Both regressions provide good fits. For simplicity, we opt for the linear model with the resulting function $time = .018 \times Excess + 3$ and $r^2 = 0.988$. This model is used in a simulation experiment in the next section to demonstrate the adaptability of this framework.

6.6.2 Simulation Experiment

To test our interaction cost model, we designed a DT system that adapts menus and demonstrate the system’s ability to make tradeoffs in simulation. For simplicity, we focus strictly on the interaction factors savings and bloat. Although these factors involve user characteristics which are not observable by the system, we assume that the system knows about the user and model the software adaptation problem as an MDP (cf. Chapter 2 for a review). In reality, user characteristics are not observable and this problem should be modeled as a POMDP instead.

We first describe the MDP model at a high level. The detailed parameters are pro-

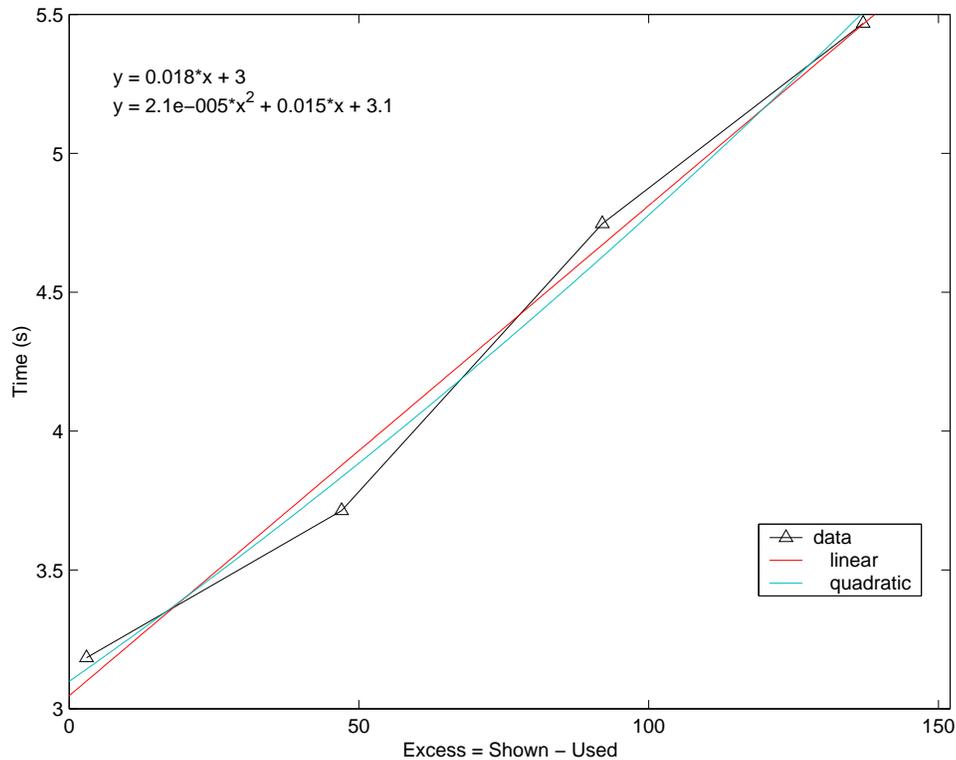


Figure 6.7: Observed and predicted average user performance for the bloat task, showing linear and quadratic regressions.

vided in Appendix B.3. The MDP models repeated interaction between the user and the system, where the system is able to take adaptive actions in the context of menu selection tasks. We restrict our attention to one menu that has a minimum of one menu item and a maximum of six menu items. The possible actions of this system are to add a menu item, delete a menu item, or do nothing. As mentioned above, the reward model in this MDP example considers bloat and savings only, although the cost of processing, interruption, and disruption also play a role in adaptive menus in a general setting. Savings is a relevant factor because the system can remove or introduce menu items that offer the user potential savings in future selection tasks; bloat is a relevant factor because the system can reduce or add to the amount of unused functionality displayed on the interface. Specifically, we define the subjective value of savings as

$V_S = f_1(\textit{Savings}, \textit{Neediness}, \textit{Distractibility}, \textit{Frustration}, \textit{Independence})$, which represents the perceived savings of the resulting interface, given the objective quality of savings and various user characteristics. In addition, we define the subjective value of bloat as $V_B = f_2(\textit{Excess}, \textit{Tolerance}, \textit{Distractibility})$, which represents the perceived bloat of the resulting interface, given the objective amount of excessive functionality in the interface, the user’s tolerance toward bloat, and how distracted the user is with more available functions. Finally, the MDP reward model is defined as $V_S - V_B$, ranging in $[-82, 82]$.

As indicated in the reward model, the MDP incorporates several user variables. These user variables are discretized as follows: *Tolerance* is binary-valued (*feature-keen* and *feature-shy* [MBB02]), and the others are tertiary-valued (three levels of frustration, neediness, distractibility, and independence, adopting our savings model from Section 4.3). With these five discrete user variables, the system models a total of 72 user types. The MDP dynamics are designed to reflect the impact that changes to the interface have on users of varying degrees of distractibility and tolerance levels. In this way, the system does not risk taking adaptive actions when dealing with highly distracted and/or feature-shy users. When an MDP is solved, we obtain a policy that maps the application and user states to an optimal action.

In the implementation, we simplify the calculation of *Excess* and *Savings* as follows. Recall that $\textit{Excess} = \textit{Shown} - \textit{Used}$, where *Shown* is the number of functions shown and *Used* is the number of functions used. With six menu items as the maximum number of functions shown in this example, we set $\textit{Used} = 1$ to roughly follow the usage pattern that only 10% of functions are used in a given interface. In addition, we approximate the calculation of *Savings* to be a value that decreases as *Shown* increases to mimic the intuition that selecting any one menu item with more items to choose from will generally take longer. (Of course, this is not always true since the system may delete a function that the user wants to select, causing the user to select the desired function somewhere, potentially further away.)

Distractibility	Tolerance	Shown	Prescribed Action
low	keen	any	add
low	shy	1	add
low	shy	2	do nothing
low	shy	3-5	del
medium	keen/shy	1	add
medium	keen/shy	2	do nothing
medium	keen/shy	3-6	del
high	keen/shy	1	add
high	keen/shy	2-6	del

Table 6.6: Solved policy illustrating the effect of bloat.

We conducted two simulation runs, both using $\gamma = 0.90$. The first one investigates the effect of bloat. For this purpose, we defined savings independent of all user variables, and focus only on bloat. A description of the solved policy is presented in Table 6.6. Generally, we see that the system deletes menu items when the number of items shown is high, except for the case of low distractibility users who are also feature-keen. For users who are highly distracted with either tolerance type, the system suggests to add menu items when one item is shown and to delete menu items otherwise. These actions result in cyclic behaviour and are likely a result of not having a cost to applying adaptive actions. We expect that adding an appropriate cost model to the system actions would resolve this issue. Note also that the values of the reward model parameters can be fine-tuned to reflect more diverse behaviour. Results from preference elicitation experiments can also be used for this purpose.

Based on this set-up and solved policy, we ran a simple simulation experiment with 100 trials, and each trial with 50 consecutive menu clicks. The simulated user selects menu items based on a Zipf distribution. The system keeps track of this usage history

as new menu selections are observed. Since the MDP policy can add, delete, or do nothing, the system uses the usage history to decide which items to add and delete. In particular, the MDP policy adds the most frequently used item that is not already in the current menu, and deletes the least frequently used item that is still in the menu. For comparison, we use a static policy that does not adapt menu items and a random policy that randomly chooses from among the same three actions that the MDP has defined.

In addition to reporting the rewards for each policy, we measured the selection time according to Fitts' Law [Fit54]. If the user tries to select an item that is not in a menu (i.e., hidden or moved somewhere else in reality), a constant amount of 1000 ms additional time is added to the recorded time. This extra time is added as a simplified way to compensate for the user's cognitive dissonance in not finding the desired menu item and then having to go to the hidden part of the menu in order to select it. These results are shown in Table 6.7. Overall, we see that the MDP policy does better than Static and Random in terms of average reward. Note that in contrast to our previous case studies, the Static menu does not always have zero reward. In previous cases, the intelligent system must tradeoff the benefits of offering help with the cost of interruption, where that cost may sometimes be greater than a static system that does nothing. Since this study focuses on interface bloat and typical function usage is about 10% of all the functions displayed, a static system that does nothing will simply accumulate negative rewards over time for users on average.

In terms of selection time, we see that average time for the MDP policy is not always as fast as the alternative policies. Since presenting a menu that does not contain the user's desired item risks a penalty of 1000 ms, we suspect this is the reason that the selection times for the MDP policy have higher averages and standard errors.

In the second simulation, we explore the system's adaptability toward various user types in the model. Here, we re-define savings as a function of all the user variables in the model in order to compare the tradeoffs made according to different user types. A

TD	Tol.		Static		MDP		Random	
			Reward	Time	Reward	Time	Reward	Time
low	keen	avg	24.0	427	73.0	427	57.4	439
		stderr	0.0	19	7.1	19	23.0	72
med.	keen	avg	-5.0	423	25.4	503	32.4	441
		stderr	0.0	20	8.9	132	23.8	82
high	keen	avg	-5.0	424	21.8	493	31.3	440
		stderr	0.0	19	9.2	127	23.8	79
low	shy	avg	-75.0	423	19.9	491	-29.3	436
		stderr	0.0	19	19.6	132	34.3	72
med.	shy	avg	-75.0	424	-25.1	490	-36.1	435
		stderr	0.0	19	9.9	133	26.4	69
high	shy	avg	-75.0	423	-25.3	507	-35.8	435
		stderr	0.0	20	9.8	129	26.4	73

Table 6.7: Results for first simulation environment, averaged over 100 trials. For convenience, TD is Distractility, Tol is Tolerance, $Reward$ is average reward, and $Time$ is selection time reported in milliseconds. The best average reward values are highlighted in bold in each case. The lowest average times are not highlighted since the static policy always has the lowest value.

description of the solved policy is shown in Table 6.8, restricting our attention to two user types only. When the user’s frustration and independence levels are low and the neediness level is high, we expect this type of user to be most receptive to the potential savings provided by added features. We define this user type as our “best case”. Likewise, we define the “worst case” user type to have a high level of frustration, low level of neediness, and a high level of independence, as we expect this type of user to perceive the least amount of savings in added features. (With all 3 distractibility types considered,

Case	Distractibility	Tolerance	Shown	Prescribed Action
best case	low	keen/shy	any	add
best case	medium/high	keen	any	add
best case	medium/high	shy	1-3	add
best case	medium/high	shy	4-6	del
worst case	low	keen/shy	any	add
worst case	medium	keen/shy	1-3	add
worst case	medium	keen/shy	4-6	del
worst case	high	keen	1-3	add
worst case	high	keen	4-6	del
worst case	high	shy	1	add
worst case	high	shy	2-6	del

Table 6.8: Solved policy illustrating the effect of user types.

note that there are 24 user types “between” the best and worst cases.) According to our MDP, for the best case user type, the system tends to suggest adding menu items because these users are receptive to adaptive help. However, when the user is highly distractible and there are already many menu items shown, the system opts to delete them. For the worst case user type, the system is more conservative and only adds menu items for a subset of the combinations of distractibility, tolerance, and number of menu items shown. Furthermore, the system deletes items when there are “too many” items shown (the actual combination depends on the user’s level of distractibility and tolerance).

Using the same simulation set-up, we compare how this MDP policy does with respect to the Static and Random policies. The results are shown in Table 6.9 for the best case and Table 6.10 for the worst case. For the best case user, we see there is very little difference in performance for feature-keen users. Note that the Random policy has higher selection times, probably due to the changes in the menu item’s relative positions since

TD	Tol.		Static		MDP		Random	
			Reward	Time	Reward	Time	Reward	Time
low	keen	avg	76.1	426	76.0	426	76.0	440
		stderr	2.1	20	2.0	20	2.0	78
med.	keen	avg	76.0	428	76.0	428	76.0	441
		stderr	2.1	19	2.1	19	2.1	73
high	keen	avg	76.0	427	76.0	427	76.0	439
		stderr	2.1	18	2.1	19	2.1	75
low	shy	avg	76.1	426	76.0	426	76.0	435
		stderr	2.0	19	2.1	19	2.0	67
med.	shy	avg	41.0	425	50.1	443	44.8	438
		stderr	2.0	19	2.9	96	4.9	74
high	shy	avg	11.0	424	30.3	442	17.7	434
		stderr	2.0	19	4.4	95	8.8	65

Table 6.9: Results for second simulation environment using the best case personality variables, averaged over 100 trials. For convenience, TD is Distractibility, Tol is Tolerance, $Reward$ is average reward, and $Time$ is selection time reported in milliseconds. The best average reward values are highlighted in bold in each case. The lowest average times are not highlighted since the static policy always has the lowest value.

items are randomly added and deleted. For feature-shy users, we see that the MDP policy makes better tradeoffs and results in higher rewards overall. Again, for this type of users, the MDP policy often deletes menu items that may be needed later. Thus, the MDP policy results in a higher selection time than the other two in comparison.

For the worst case user, we see there is very little difference in performance for low distractibility users (although the Random policy again has higher selection times here). For the other user types, we see that the MDP policy makes better choices and receives

TD	Tol.		Static		MDP		Random	
			Reward	Time	Reward	Time	Reward	Time
low	keen	avg	69.9	428	70.0	428	70.0	440
		stderr	2.0	19	1.3	19	2.1	75
med.	keen	avg	15.0	425	58.2	444	34.4	441
		stderr	2.1	18	9.2	96	19.8	80
high	keen	avg	5.0	423	52.9	444	24.2	437
		stderr	2.0	19	10.1	97	22.1	69
low	shy	avg	69.9	425	69.9	425	70.0	440
		stderr	2.1	19	2.0	19	2.0	75
med.	shy	avg	-20.1	427	-5.6	445	-14.1	440
		stderr	2.0	19	3.6	96	6.8	75
high	shy	avg	-80.0	424	-34.0	484	-73.9	434
		stderr	2.1	19	13.9	140	11.6	70

Table 6.10: Results for second simulation environment using the worst case personality variables, averaged over 100 trials. For convenience, TD is Distractility, Tol is Tolerance, $Reward$ is average reward, and $Time$ is selection time reported in milliseconds. The best average reward values are highlighted in bold in each case. The lowest average times are not highlighted since the static policy always has the lowest value.

higher reward averages overall. While usability studies are needed to confirm the initial simulation results, these simulation results suggest that our system is able to make appropriate tradeoffs between the cost of bloat and the benefits of savings for various user types.

6.7 Eliciting the Subjective Value of Help

As discussed in previous chapters, considerable work has been devoted to the prediction of user needs and goals (e.g., [HBH⁺98, AZN98, Les97, BA04] among others), much of it is focused on developing probabilistic models of user goals. Less emphasis has been placed on assessing user preferences for software interaction and customization (for exceptions, see [GW05, HKA04]). However, accounting for user preferences is critical to good interface customization. For instance, consider automated word completion [FLL02b]. Some users may prefer single-word suggestions, while others may prefer several different suggestions. Similarly, some users may be satisfied with “partial help” (e.g., a partially correct word that saves a few keystrokes) while others may wish to use only completely correct completions. These preferences, and more importantly, a user’s *strength of preference*, are needed to make suggestion decisions: these preferences must be weighed against the probability of specific user goals.

In this section, we evaluate the effectiveness of a variety of preference elicitation techniques for interface customization [HB08]. While elicitation may not be used directly (i.e., online) during application use, most adaptive systems will make indirect assessments of user preferences (e.g., [SL01, MC06, CLWB01, HB06b]). However, even indirect assessment methods require some knowledge of the range of possible user preferences and how those are (perhaps stochastically) related to observable behaviour. In these cases, offline preference elicitation for different customizations can provide valuable data for the design of an online system.

Most existing literature on preference elicitation in AI, as well as the majority of that in behavioural decision theory, assumes that people “know” their preferences *a priori*. However, often users have not encountered, nor even considered, the hypothetical situations typically posed in an elicitation process. This is especially true of software adaptation, since the situations involve novel interfaces. Under these circumstances, people may report their *predicted utilities* by conceptualizing the posed scenarios and

forecasting their own preferences. However, what people “think they like” can systematically differ from what they “actually like” [KS90]. For example, someone who has not actually engaged in a system that offers a range of partial word completion suggestions may predict that they dislike the interface in a particular circumstance, but in fact like it when they experience it (or vice versa).

For this reason, we propose a novel *experiential elicitation* approach for interface customization. Our elicitation “queries” allow users to assess *experienced utilities* [KWS97] by providing simple, hands-on tasks and system suggestions or customizations, drawn from a particular distribution. We adapt standard elicitation approaches to use such *experiential queries*. Our approach also overcomes some of the difficulties with well-established procedures (e.g., standard gambles) that involve probabilities over a distribution of outcomes. We explore this new approach in the context of a specific customization task — suggesting highlighting options in PowerPoint — and show that experiential elicitation offers a more accurate means of assessing quantitative tradeoffs in preferences. One drawback of experiential queries is the time they take. To counteract this, we also propose two hybrid models that attempt to assess experienced utilities somewhat more (time) efficiently. Our results show that a hybrid procedure provides a good approximation to the experiential approach in a much more effective manner.

6.7.1 Highlighting Task in the Customization Domain

We are interested in developing intelligent systems that perform online interface customization based on user preferences and user needs. We focus on a concrete form of customization as an example to illustrate our approach, but note that the general principles apply more broadly.

Recall the PowerPoint highlighting goals developed in Chapter 5. A user wishes to highlight important phrases and new terminology by applying a particular font style. For aesthetic reasons, the user tends to choose from just a few highlighting styles that are

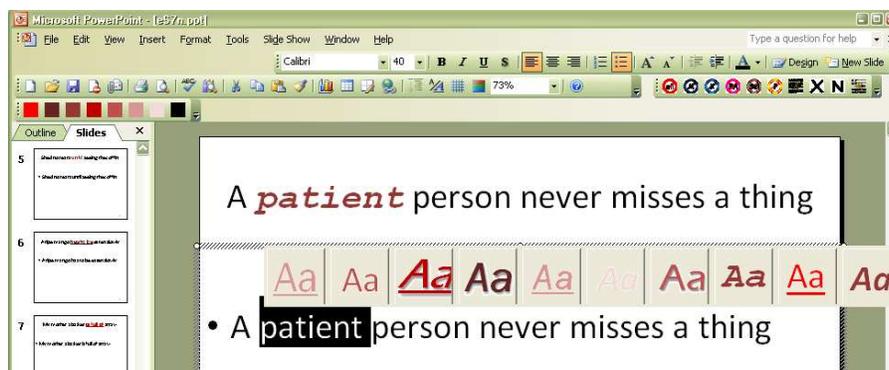


Figure 6.8: Icon suggestions to help the user in a highlighting task implemented as part of PowerPoint 2003.

consistent throughout a presentation, and possibly across multiple presentations. This consistency provides the opportunity for an intelligent system to observe and learn user-specific patterns, so that it can offer useful suggestions in the future. Figure 6.8 shows an example toolbar of 10 icons, each suggesting a particular set of font attributes that can be applied to highlight the selected phrase “patient.” Many repetitive tasks in PowerPoint (e.g., those described in Section 5.1) and other software can also benefit from automated suggestions designed to minimize user effort.

More precisely, we assume that in a highlighting task, the user has a certain style in mind, which involves changing some number of font attributes to make it stand out from the rest of the slide. We define the *complexity* of a highlighting style as the number of font attribute values required to create it. In Chapter 5, we referred to this parameter as K . For example, relative to “plain” black text, a bold, italicized font has complexity $K = 2$.

To assist the user with the highlighting task, our system can suggest a toolbar with a number of icons, each offering some combination of font attributes shown by the characters “Aa” (as illustrated in Figure 6.8). The user may select one of these icons to apply the associated font style; complete the highlighting task purely manually using the mouse (e.g., click on the Bold icon) or shortcut keys (e.g., press Ctrl-B); or accept a suggestion,

but further refine it manually by applying (or undoing) additional font attributes. If the toolbar is ignored (e.g., the user continues typing), it disappears after a short time.

6.7.2 The Value and Costs of Suggestions

As introduced in Section 6.1, various interaction factors are relevant to toolbar suggestions. In this experiment, we focus our attention on two interaction factors — savings and information processing — and one user characteristic — neediness. The quality of help actions in adaptive systems can be defined to capture the difference between manual effort required by the user with and without help. We define the quality $Q(i|g)$ of a suggestion icon i (given a goal g) to be this difference. For simplicity, we assume users to pick the best icon available. Thus, we define the quality of toolbar t to be $Q(t|g) = \max_i Q(i|t)$, the maximum effort saved by any icon in the toolbar. Note that since the goal is only known stochastically, expected quality must be computed relative to the system’s beliefs about the user’s current highlighting goal.

As we discussed in Section 6.3.3, various user characteristics impact the subjective value of savings. In our initial design of the elicitation procedure, we focus on neediness only. Examples of observable characteristics that a system can use to infer the user’s neediness level were discussed in Chapter 4, which include the user being stuck (i.e., pausing from task activity) or looking for help (i.e., browsing without selection). These observations often arise when the task is difficult to accomplish. Therefore, we define the user’s level of neediness $N(g)$ as a function of how difficult the goal g is to the user. In Section 6.7.5, we explain how we simulate the user’s level of neediness in a controlled experiment.

Apart from potential savings, we model the information processing cost associated with toolbar suggestions. In adaptive toolbars, the icons that appear as well as the order in which they appear are always changing. Thus, users cannot develop a search strategy to minimize processing time. For simplicity, we do not model the user’s speed in this

setting. To model processing time, we focus only on the number of items in an adaptive toolbar. We define $L(t)$ as the number of icons in t .

6.7.3 Brief Review of Preference Elicitation

In order for the system to choose a good toolbar to help the user, it needs a model of the user’s preferences over the possible attributes. The value of a suggestion t is a function of the user’s utility function with respect to N, L, Q . We define the set of possible outcomes O over the range of values of these three attributes. We use notation such as $n1, l5, q4$ to represent the outcome with the user’s neediness level at 1 (high) and the toolbar showing five icons with toolbar quality 4. For example, Figure 6.8 shows the interface outcome for $n1, l10, q4$, since the interface is simulated for the neediness setting (as we explain below), the toolbar displays ten icon suggestions, and the macro that matches all four font attributes of the target highlighting pattern is present in the toolbar.

A user’s preferences for particular outcomes, including their strength of preference, can be represented by a *utility function*, $u : O \rightarrow \mathbb{R}$, such that $u(o_i) > u(o_j)$ iff o_i is preferred to o_j , and $u(o_i) = u(o_j)$ iff the user is indifferent between o_i and o_j . For convenience, we normalize utilities to the interval $[0, 1]$, defining o^\top to be the best outcome with $u(o^\top) = 1$ and o_\perp to be the worst outcome with $u(o_\perp) = 0$. A utility function can be viewed as reflecting qualitative preferences over *lotteries* (distributions over outcomes) [KR76], with one lottery preferred to another iff its expected utility is greater. Let $SG(p) = \langle p, o^\top; 1-p, o_\perp \rangle$ denote a *standard gamble*, a specific (parameterized) lottery where o^\top is realized with probability p and o_\perp is realized with probability $1 - p$. The expected utility of this lottery is p .

The *standard gamble query* (SGQ) for outcome o_i asks the user to state the probability p for which he would be indifferent between $SG(p)$ and outcome o_i [KR76]. This type of query is extremely informative as it asks the user to assess a precise tradeoff involving o_i , and indeed fixes $u(o_i)$ on the normalized scale. However, this makes such queries

Query Type	Question	Range of User Reponse
$SQG(o_i)$	For what value of p are you indifferent between $SG(p)$ and o_i ?	$p \in [0, 1]$
$Bound(o_i, p)$	Given p , do you prefer $SG(p)$ to o_i ?	Yes or No

Table 6.11: A summary of two types of queries.

practically impossible to answer with confidence. More cognitively plausible are *bound queries*. The bound query $B(o_i, p)$ asks the user whether he would prefer $SG(p)$ to o_i . A positive response places an upper bound of p on $u(o_i)$, while a negative response places a similar lower bound. Their yes/no nature makes bound queries easier for people to answer. A summary comparison of these two types of queries is shown in Table 6.11.

In principle, bound queries can be used to incrementally elicit utility functions to any required degree of precision by incrementally refining the bounds on utility outcomes, at each stage giving rise to a more refined set of *feasible* utility functions (those consistent with the bounds). From a procedural perspective, for a given outcome o_i , the experimenter repeatedly poses a series of bound queries $B(o_i, p)$ with varying values for p until the user's response indicates that he is indifferent between the two options of the query. At that point, we have $u(o_i) = p$. On the other hand, the experimenter adjusts p according to the user's response. In particular, given query $B(o_i, p)$, if the user's response is "Yes", then $u(o_i) < p$ and we decrease the value of p in the next query. Similarly, if the user's response is "No", then $u(o_i) > p$ and we increase the value of p in the next query. By choosing a new value of p that divides the remaining feasible utility region by half, this procedure mimics a binary search in the utility region of an outcome. In practice, as the feasible regions for each parameter become small, the queries will generally become harder to answer with confidence.

In practice, a lottery is presented as textual (or verbal) description of two outcomes and their probabilities, possibly accompanied by visual aids representing the outcomes.

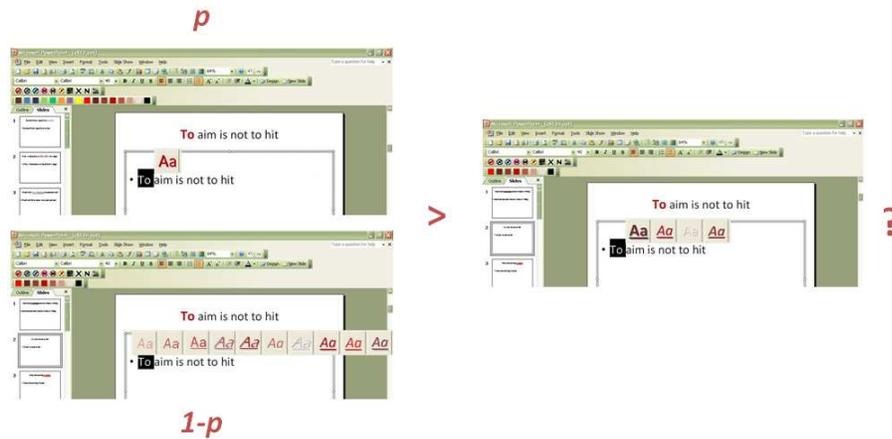


Figure 6.9: Printed screen shots used in a bound query to help users visualize the standard gamble option (shown on the left hand side) and the fixed outcome option (shown on the right hand side). The printed screen shot for the fixed outcome varied according to the particular N, L, Q values for the conceptual bound query of interest.

We refer to this delivery of bound query as a *conceptual query*, since people are asked to think about the two alternatives before making a decision. As an example, Figure 6.9 shows the use of printed screen shots used to represent a particular conceptual bound query. During conceptual elicitation, each query involves asking the user to think about the two options by imagining the use of two separate systems to complete the highlighting task. $SG(p)$ corresponds to using an adaptive system repeatedly, with p percentage of the instances involving the best interface o^\top and the rest o_\perp . Option o_i corresponds to using a static system with interface o_i repeatedly.

Though theoretically appealing, people often have difficulty assessing the probability parameter in SGQs [KR76]. Similarly, bound queries with precise probabilities can be hard to conceptualize in this domain, with people having difficulty comparing interface “lotteries” with a fixed outcome. Finally, a user’s assessment of what they like in hypothetical settings can differ systematically from what they actually like [KWS97, KS90, KT06, HZ04]. For these reasons, we investigate alternative elicitation

mechanisms that overcome these difficulties.

Preference elicitation in interface customization has typically adopted a qualitative approach to assessing user preferences that learns preference rankings without learning the strength of those preferences [LHL97, PFT03, GW05] (although Horvitz et al. [HKA04] uses “willingness to pay” to quantify the cost of interruption). However, given the stochastic nature of goal estimation, we require estimates of utility as discussed above. To do this within our POMDP-DAISI framework, we require some means of associating observed user behaviour with utility functions. While direct online elicitation is not (completely) feasible, offline elicitation can be used to develop the required models. For instance, utility functions elicited offline can be clustered into a small set of *user types* [CGNS98], which the intelligent system assesses online. Online assessment of user types may be done passively [HB06b] or explicitly through active elicitation [Bou02]. In this context, we propose to use an experiential elicitation procedure to carry out offline elicitation experiments with real users, which we describe in the next section.

6.7.4 Experiential Elicitation for Interface Customization

To facilitate interpretation of bound queries, we develop an *experiential* version of bound queries which allows the user to “experience” both query options (including the stochastic one) before stating a preference. The user is asked to complete k simple tasks using the system in each of two ways: the $SG(p)$ option shows o^\top in fraction p of the k tasks and o_\perp in the remaining tasks (in random order). In this way, the user “experiences” a stochastic mixture of interfaces. The deterministic option also requires k task completions but all using the same interface o_i . After each option, the user is asked to reflect on what he liked and disliked about the experience — which could be a function of the effort required by the tasks, toolbar “processing” cost, the satisfaction in having toolbar help available, or the ease of interaction — and to indicate his preference. The response provides a bound at p for o_i .

We elicit $U(N, L, Q)$ from users using the interface shown in Figure 6.8. We use $k = 10$ tasks for each alternative in the query, so that the probability parameter p in the standard gamble is coarsely discretized into ten equal sized regions, i.e., $p = 0, 0.1, 0.2, \dots, 1.0$. With two options, each query involves the user completing 20 highlighting tasks in total. To reduce the cognitive burden and experiment time, we elicit only $U(N, L, Q)$ for the values $Q \in \{0, 2, 4\}$, $L \in \{1, 5, 10\}$, and $N \in \{0, 1\}$. This discretization yields a range of 18 outcomes. Similarly, we discretize query probabilities: $[0, 0.1, 0.2, \dots, 1.0]$. We define $o^\top = n0, l1, q4$, since the user is at a low level of neediness and receives the best help possible, and $o_\perp = n1, l10, q0$, since the user is at a high level of neediness and receives the worst help possible.

When the system suggests a toolbar, the user can select one icon or ignore it altogether. If the wrong font style is chosen, or the suggestion is ignored, the user must carry out the highlighting task manually. In contrast to the conceptual condition, choosing a suggestion icon that does not match the target goal perfectly implies extra effort for the user. To summarize, each task requires the user to carry out several separate actions, and applying an incorrect style requires additional fixes.

6.7.5 Conceptual versus Experiential Elicitation

We compared experiential and conceptual queries experimentally to investigate their impact on elicitation based on several criteria: the efficiency (duration) of the procedure; the cognitive demand imposed on users; the interpretability or understandability of queries by the user; and the quality of the elicited responses. In particular, we compare the elicited utilities quantitatively under these two conditions, and test whether mean utilities elicited under conceptual elicitation are the same as those under experiential elicitation (our null hypothesis, H_0). We are also interested in the general structure of $U(N, L, Q)$, specifically, if users perceive any value in such simple help, and if any trends across the user population can be seen in the underlying preferences.

In addition to verbal descriptions, we used screenshots to represent each outcome in the Conceptual condition. Thirteen people participated in the Conceptual condition and 8 people in the Experiential condition.

To control for the user’s highlighting goal, we define a *target font style* in each task. As illustrated in Figure 6.8, each PowerPoint slide has two sentences with the same words, where the top sentence indicates the phrase highlighted with the target font style. The user’s highlighting task is to match the first sentence by changing the font style of the appropriate words in the second one. The vocabulary of target font styles is defined by 7 features—5 of which are binary (bold, underline, italics, shadow, size increment) and 2 of which are multi-valued (colour has 8 values, font family 10). All target font styles in the experiment have complexity 4. Therefore, if the system’s suggestion is perfect, it can save the user from manually executing (sequences of) 4 separate events.

Recall that neediness is a hidden user variable, and how much help a user needs is a function of how difficult the user’s goal is to accomplish. In the experiment, we simulate two neediness settings by controlling the task environment, and thus, making the user goal more difficult to achieve. To simulate a needy user ($n1$), we make the task more difficult by restricting the set of colours to 7 shades of red and the set of font families to 4 similar fonts. In this way, the system’s feature vocabulary, the target font styles in the highlighting tasks, and the icons in the toolbar are restricted to similar colours and fonts. The interface in Figure 6.8 shows an example of this neediness setting. The default feature vocabulary (and hence, possible target font styles and toolbar icons) defines the interface environment for a user who is not needy ($n0$).

During the elicitation, we posed bound queries and incrementally refined the bounds by choosing p to be the midpoint of the set of feasible utility functions until all the outcomes have feasible regions ≤ 0.1 . To illustrate, Figure 6.10 illustrates sample results from a specific elicitation run, showing a partial utility function (for fixed values of N and L) for three users. The elicited bounds are drawn as error bars (note that discretization

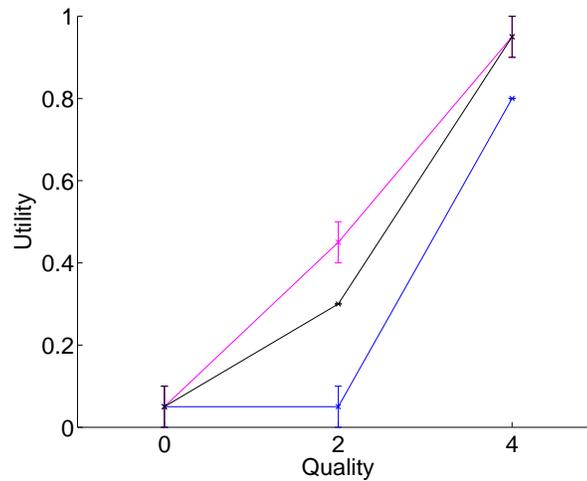


Figure 6.10: Partial utility functions for $n0, l10$ as a function of Q elicited from 3 users in the Experiential condition. Lines are drawn at the midpoints of the resulting bounds.

prevents us from pinning down the utility function precisely). The “blue” user (lower line), for instance, has $0 \leq u(n0, l10, q0) \leq 0.1$. The bound at $q4$ for the blue user is tight because he is indifferent between $SG(.8)$ and $o_i = n0, l10, q4$.

6.7.5.1 Methodological Comparison

On average, each experiment took 30 minutes in the Conceptual condition and 2 hours (divided into two sessions) in the Experiential condition. Experiments in the Experiential condition are much longer because experiential queries require users to carry out tasks, while the conceptual queries only require users to think about the scenarios.

Since experiential queries require a series of task completions, users became tired early on and found it necessary to take breaks in order to not be confused with the various options and associated experiences. Users in the Conceptual condition did not seem tired during the procedure, but they were at times inconsistent with previous responses.

Although experiential queries took longer, they provided hands-on experience and therefore required little verbal explanation. In contrast, conceptual queries were often

difficult to explain, because they require users to first understand the aspects of the interface (e.g., amount of effort needed in manually completing the tasks, controlled quality of help in the suggestions), then compare the costs and benefits of a mixture of interfaces with a definitive interface, and finally, imagine using the respective interfaces in a repeated scenario.

6.7.5.2 Structural Comparison

Independently of the elicitation method, we are interested in the perceived value of this adaptive form of customization. The utility functions across the 21 users varied widely — some are convex, some are concave, some are linear, some are “flat” when quality is not perfect, and some are “flat” when length is not one. Some examples (using midpoints of feasible regions) are shown in Figure 6.11. Clearly, user preferences vary widely, even for such simple highlighting help with three customization attributes.

In general, the utility functions are monotonically non-decreasing in Q when N and L are fixed, and monotonically non-increasing in L when N and Q are fixed. In particular, when help quality is high ($q4$), utility decreases slightly as L increases. This is expected as users perceive higher processing costs with more icons. We also see that *partial help* ($q2$) with L at $l1$ is qualitatively different than $l5$ or $l10$, because more icons decrease the chance of the user identifying the single icon that provides partial help. When help quality is low ($q0$), some users prefer to see one bad suggestion ($l1$) than many bad suggestions ($l10$). There are no general trends in utility given neediness; some users showed clear differences between the needy ($n1$) and not needy ($n0$) scenarios, while others viewed them the same. From this, we see that users perceive value in automated help, even in simple tasks such as highlighting. Of course, more data is needed to draw definitive conclusions about possible parametric forms for utilities that could further simplify online assessment.

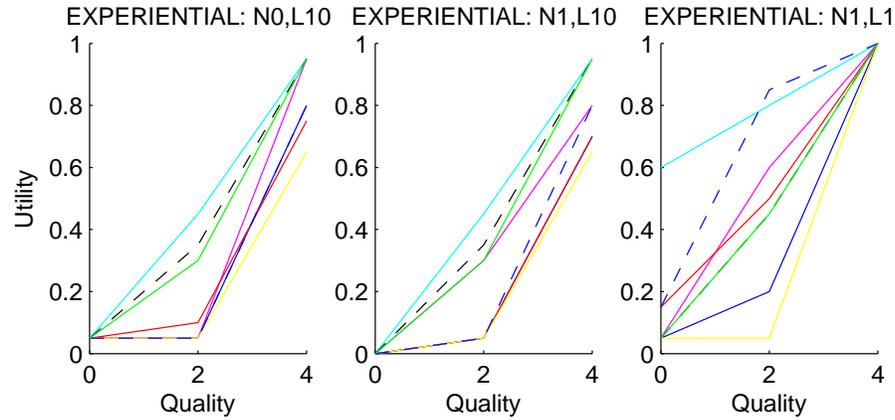


Figure 6.11: Partial utility functions for $n0, l10$ as a function of Q elicited from all 8 users in the Experiential condition. Each line connects the midpoints of the outcome’s elicited bounds for each user.

6.7.5.3 Quantitative Comparison

Using Hotelling’s T^2 statistic,¹ we found that the mean utilities in the two conditions are significantly different, $p < 0.01$. Thus, we reject H_0 . To provide a more detailed comparison, we carried out a two-tailed t -test with independent means for each outcome. The resulting t scores² are plotted in Figure 6.12. We use this plot as a way to give us more insight into the user responses across the two conditions.

Interestingly, Figure 6.12 shows that the mean t score tends to be higher for most outcomes in the Experiential condition, including outcomes that provide partial quality help ($q2$) and incorrect suggestions ($l5, q0$ and $l1, q0$). This indicates that users in the Experiential condition perceive greater value in adaptive help than users in the Conceptual condition. The Experiential condition requires users to carry out 20 tasks for each query, while the Conceptual condition only asked users to “think about” the tasks. With

¹The T^2 distribution is a multivariate analog of the Student’s t -distribution. We used the Moore-Penrose pseudoinverse in computing T^2 . Also, we assume independence, normality, and homogeneity of (co)variances in our analyses. However, these assumptions remain to be tested. Thus, our results provide suggestive evidence only.

²A t score is a measure of how far apart the two sample means are on a distribution of differences between means.

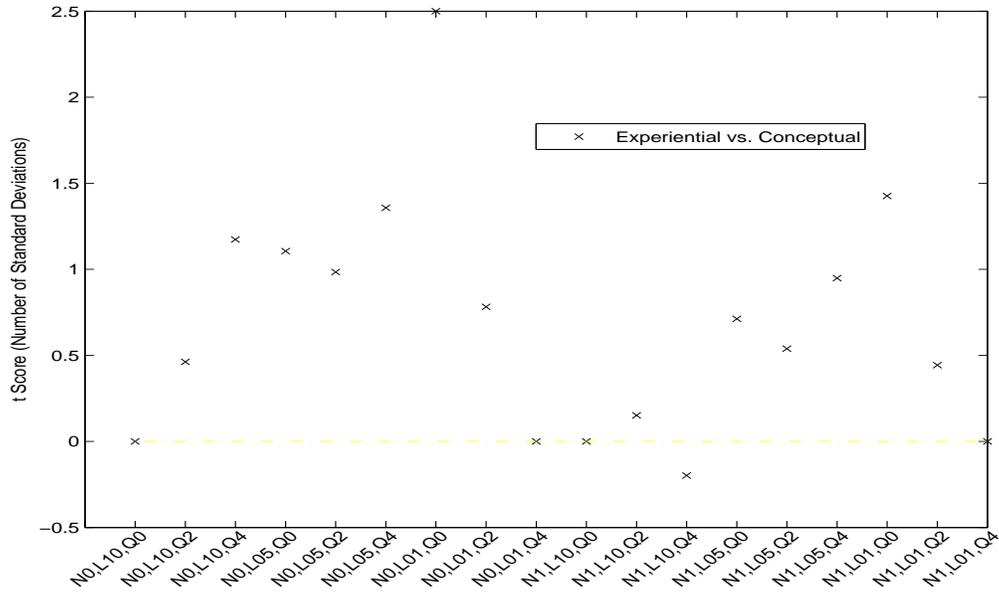


Figure 6.12: Resulting t scores for each elicited outcome.

conceptual queries, we believe that participants are less likely to truly perceive the value of automated help in repeated scenarios, and thus, underestimate the utility of these outcomes.

6.7.6 Ways to Improve the Experiential Elicitation Procedure

Although experiential queries enable users to report more realistic preferences, the procedure is time consuming (even with simple utility functions over 18 outcomes). We develop two more efficient elicitation procedures, based on the findings in Section 6.7.5. Following the same experimental set up, we introduce two procedures, *primed* and *primed+*, that attempt to elicit experienced utility more effectively. The Primed condition uses a training session to familiarize users with the interface and the attributes N, L, Q ; but the elicitation procedure itself still relies on conceptual queries only. The Primed+ condition uses this training session plus 5 experiential queries at the start of the elicitation. The remaining elicitation is done using conceptual queries only.

Similar to the previous experiment, we want to test whether the mean utilities elicited under the Conceptual condition are the same as those elicited under the Primed and Primed+ conditions (our null hypothesis H_0). A total of 9 and 8 people participated in the Primed and Primed+ conditions respectively.

6.7.6.1 Methodological Comparison

Both procedures were easier to administer than the experiential and conceptual ones. First, the familiarity acquired in the training session reduced the need to explain the interface to the users. On average, each experiment took 30 minutes in the Primed condition and 60 minutes in the Primed+ condition. Neither condition seemed tiring for users. Second, users in both conditions found the queries easier to understand than users in the Conceptual condition. Finally, users in the Primed+ condition were able to use their experiential query responses as a reference for future responses. These initial experiential queries provided users with a quick feeling for the sequential costs and benefits of using the toolbar.

6.7.6.2 Quantitative Comparison

We conducted a pairwise analysis between the mean utilities in the Conceptual and Primed conditions, and between those in the Conceptual and Primed+ conditions. Using Hotelling's T^2 statistic, we found that the mean utilities between Primed/Primed+ and Conceptual conditions are significantly different ($p < 0.01$ and $p < 0.05$ respectively). Thus, we reject H_0 in both instances. The results of a pairwise analysis using a two-tailed t -test with independent means for each outcome are shown in Figure 6.13.

From Figure 6.13, we see that the primed means are generally lower (t score less than 0) than conceptual. In fact, the t scores for the Primed condition (vs. Conceptual) are always lower than the t scores for Experiential (vs. Conceptual). One explanation for this is the fact that the training session in the Primed setting gave users a quick

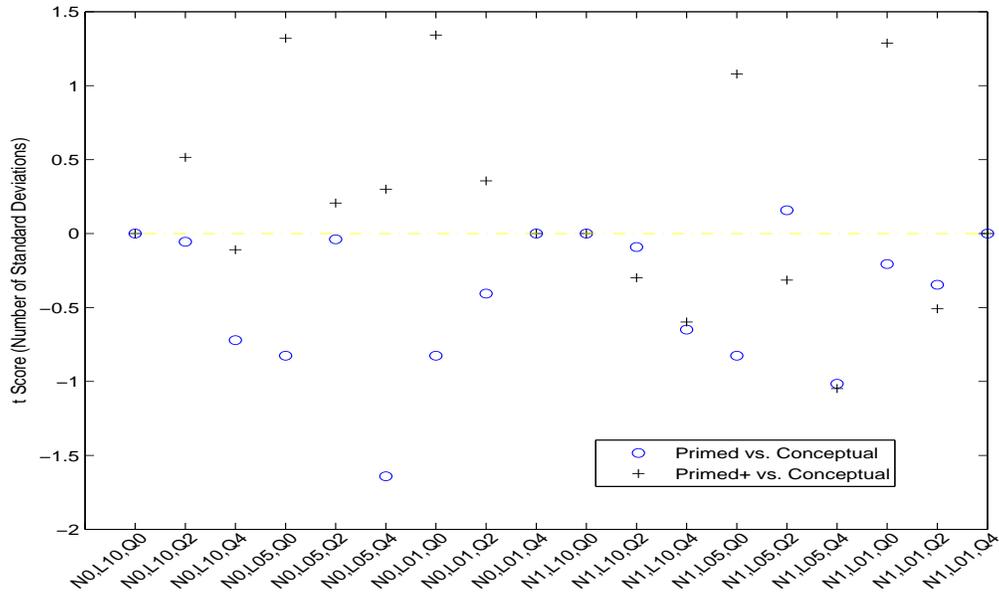


Figure 6.13: Resulting t scores for each elicited outcome.

estimate of the costs of searching through and evaluating suggestions in the toolbar — a cost that would otherwise be “unknown” in the Conceptual condition — but that the short experience was insufficient to provide the user with a sense of the *benefits* of help. In both the Primed and Conceptual conditions, the cognitive demand involved in repeated highlighting tasks and the value of help is consistently underestimated. In contrast, the Primed+ condition and its five experiential queries provide a sense of long-term benefits of using the toolbar. Indeed, in Figure 6.13, the general pattern indicates that Primed+ approaches Experiential. These results support our initial hypothesis that experiential queries enable the user to perceive the full value of adaptive help under realistic circumstances.

6.8 Summary

In this chapter, we described a set of interaction factors that account for the costs and benefits involved in actions taken by intelligent software adaptation systems. These

factors include information processing, target selection, visual occlusion, bloat, savings, disruption, and interruption. We proposed a generalized additive utility structure using these factors to model the objective and subjective value of interaction. Where available, we adopted existing formal models for these factors. In the case of occlusion, bloat, and disruption, we designed data collection experiments to learn appropriate models. Furthermore, using the learned model for bloat, we demonstrated in Section 6.6 how it can be used in an MDP model to construct an adaptive menu. The focus of this case study was to better understand the impact of modeling bloat in this simple set-up. We found that the prescribed system action adapts to different levels of tolerances to interface bloat, as well as adapting to other user characteristics such as distractibility, frustration, independence, and neediness. Lastly, in Section 6.7, we introduced an experiential procedure for eliciting the subjective value of help using the PowerPoint highlighting environment developed in Chapter 5. In comparison to a conceptual elicitation procedure, we found significant differences in the perceived utility of automated help elicited from users under the two conditions. This finding supports existing results in psychology that indicate differences between what people think they like versus what they actual like, given experience with the alternatives in the environment. While we believe our experiential procedure is better able to elicit the more accurate perceived values of automated help, it is too time consuming and cognitively demanding to deploy. As a result, we modified the procedure to make it faster but still allow users to assess their experiential utilities. Overall, we identified and formalized a set of interaction factors for intelligent software adaptation systems and demonstrated the development of the Action Selection Component within the DAISI framework. Future work should focus on the integration of the developed models and proposed elicitation technique in more complex decision making scenarios.

Chapter 7

Conclusions and Future Work

In this thesis, we explored the use of a decision-theoretic framework and its ability to support the modeling of a broad range of user types, and the inherent uncertainty associated with this, in the domain of intelligent software adaptation. The main contribution of this thesis is the development of the DAISI framework, with the main emphasis placed on the design of the user characteristics model and its impact on the utility function. In this context, a general model for the system's utility function was proposed (Figure 6.1). This model is designed to support modifiability so that different subutility functions can be chosen or eliminated from the intelligent system's reward model as needed per application. To demonstrate this framework, we implemented and evaluated several prototypes, with each prototype emphasizing different aspects of the model (presented in Chapter 4, Chapter 5, and Chapter 6). Overall, we have shown that it is possible to develop an intelligent software adaptation assistant that recognizes and adapts to a broad range of users, and that this value is recognized by users.

In general, a central question of interest in model development is how a designer should choose between using a rich, realistic model (coupled with approximate inference as needed) versus an approximate model (which may use exact or approximate inference depending on the granularity). First, we saw empirical evidence that users accept au-

tomated help that is partially correct and manually fix the incorrect part afterwards. This observation suggests that perfect accuracy of the user goal model is not required for user acceptance. While we believe that suggestion accuracy and the user's perceived utility of the system are related, it seems that goal accuracy beyond a certain point plays less of a role in perceived utility than other factors, such as user characteristics and the cost of suggesting the wrong help for certain goals or under certain interface settings. Thus, we hypothesize that development efforts in improving goal inference accuracy will offer diminishing returns beyond a certain level, although the specific identification of an appropriate level of accuracy requires further investigation. Second, we saw empirical evidence that different users perceive the same system actions differently, even when presented with the same interface and interaction parameters. This observation suggests that models without a subjective utility component cannot be used to derive appropriate help actions or suggestions, thus, limiting broad user acceptance of the system. Moreover, a system that focuses purely on model accuracy and less on the action selection component is less able to account for subjective utility. In this thesis, we modeled subjective utility via a set of user characteristics. Further investigation is needed to better understand the relationship between the accuracy of inferring user characteristics and perceived system utility.

The process of developing and evaluating our approach has indicated some difficulties as well as interesting paths for future exploration. In light of this experience, we summarize the lessons learned and future work below.

7.1 Developing the User Model

Throughout our user studies, we found that different users prefer different ways of interacting with the system. We emphasize the importance of intelligent systems being able to adapt their interactivity level to users who welcome help as well as to users who

are unwilling to consider automated assistance regardless of the quality. In our work, we approached this problem by building a system that recognizes different user types and tailors its behaviour to different user states and types. Complementary work that explores mechanisms to make the system more transparent, to gain user trust, and to provide explanations can be incorporated as well [KKL02, TDCF07, BJJA05, BMC07].

In this thesis, we have adopted a model-based approach that explicitly models user characteristics. With respect to development efforts, this approach makes the system more transparent and easier to understand. In an attempt to collect labeled data, we found that some user characteristics, such as neediness and distractibility, were relatively amenable to constructed proxies for these unobservable variables. For example, pop-up boxes and animation can be used in controlled conditions to make users more distracted, and the interface and task can be designed to make users more needy. On the other hand, labeled data on user characteristics such as frustration and concentration were more difficult to collect. Other types of observations, and ways to facilitate the collection of labeled data, need to be explored. While an explicit model of user characteristics has its advantages, the cost of collecting labeled data and increased difficulty in validation also need to be considered.

Although user control has been suggested in the literature to be an important criteria in the design of adaptive systems, it is unclear how intelligent systems can be used to customize software while leaving users in control. In one study where we included a slider widget that allows users to control the interactivity level of system suggestions showed that none of the users attempted to use this widget. In other studies where a user control widget was not provided in the interface, none of the users expressed the desire for or inquired about a manual customization alternative. Our initial observations suggest that people rarely initiate configuration (admittedly, in a very controlled setting), and that preference elicitation and system initiated customization methods are needed.

It is important for the observation model to be able to assess potential causes or

intent underlying the rejection of help. For example, when a user ignores an automated suggestion, it could mean (at least) one the following: (i) the user likes automated assistance in general, but is ignoring the current suggestion because his attention is somewhere else; (ii) the user likes automated assistance in general, but is ignoring the current suggestion because the suggestion is incorrect (but has a low cost associated with it); (iii) the user likes automated assistance in general, but is ignoring the current suggestion because of the type of tasks he is temporarily engaged in; (iv) the user dislikes automated assistance in general, and is ignoring the current suggestion because he really does not want any help from the system. In all these cases, the system observes that its suggestions are being rejected. Without additional evidence, this information alone would lead a system that adapts its behaviour to offer suggestions less often over time. However, in cases (i-iii), such behaviour is not necessarily warranted, and need to be carefully differentiated from case (iv). Use of aggregate information to indicate user's attitude toward acceptance of help may be used to identify cases (i) and (ii). To help distinguish case (iii), additional information that associates user acceptance with the type of help being offered, or to the type of goals being pursued, may also be used. These ideas need to be investigated further.

7.2 Policy Design

As we saw in user feedback in several of our experiments, even when the automated help provided by the system does not match their goals perfectly, some users will accept help knowing that they have to manually fix part of the suggestion afterward. This behaviour suggests that at least some users find partially correct help useful, although how users tradeoff the utility of partially correct suggestions and the cost of fixing the accepted help needs to be investigated further. In contrast to the findings from the Lumière project, giving wrong suggestions is not necessarily costly. We suspect that the nature of the

automated assistance contributed to this difference. In the case of Lumière, users were given advice to solve a problem, such as how a formula can be calculated in Excel. From the user's perspective, the steps are more involved and time consuming in these types of problems. Thus, taking a wrong step in the suggestion or figuring out which part of the advice has to be fixed may require more cognitive effort and time on the part of users than in the test applications we considered in this thesis. As proposed in our work, costs in terms of time and cognitive effort should be quantified and incorporated into the system's utility function. We believe that future studies should be done to better understand the precise cost models involved in adaptive systems that offer different types of help.

In situations where users are known to accept partially correct help, the use of an intelligent selection strategy that provides better coverage of possible help actions in a single, joint action is recommended. In particular, if the system is able to choose a set of, say J , actions to suggest to the user, rather than choosing the J most likely actions, we recommend choosing the J most *useful* actions (taken in expectation with respect to the user goal and user characteristics distributions). The design of such an action selection strategy was illustrated in several cases (c.f. Section 4.3, Section 4.4, and Section 5.4). The rationale for this recommendation is that choosing a set of suggestions based only on a distribution will result in suggestions that are similar to each other, while using expected utility given such a distribution will allow more diverse suggestions to be made. Therefore, in a context where users are willing to accept partially correct help, a strategy that provides more options for users to choose from is more likely to result in increased user acceptance.

Based on informal user feedback from Section 5.5, we found that simple policies that were not able to adapt their suggestions incrementally gave users an immediate impression of poor performance. For example, a frequency policy that only updates its distributional information after goal completions offers poor help in sequential tasks

such as highlighting. We conjecture that, all else being equal (e.g., accuracy, presentation manner), the ability of a policy to adapt its suggestions incrementally will have higher utility than one that only updates its suggestions after a goal completions. To our knowledge, we are unaware of any HCI literature that investigates the utility of adaptive policies in sequential tasks. (For example, the most popular task for comparing adaptive policies is menu selection, which is not sequential.) This hypothesis should be tested further in future experiments. It would be interesting to explore the structure of perceived utility for simple, non-incremental, adaptive policies as a function of accuracy.

In our case studies of a text editor application with word prediction help and PowerPoint with highlighting help, we employed myopic policies in the action selection component. Our myopic policies consider the expected value of a help action, given the system's belief that the user will accept an action when it is offered. In these applications, the goals of typing a word and highlighting a phrase has a very short horizon; the impact that system actions have on the user is limited. While the use of myopic policies may be reasonable in such settings, we recommend the use of long term policies when the problem horizon is larger. For example, we employed long term policies in constructing adaptive menus that are able to add and hide menu items as help actions (cf. Section 4.4 and Section 6.6.2). The impact of an added or hidden menu item has long term effects on all future usage of that interface. The case study in Section 4.4 demonstrated the design of a policy that weighted the benefits of expected long term savings in overall selection time by adapting the location of a menu item, against the cost of expected disruption to the user's mental model induced by that system action. With emphasis on bloat, the case study in Section 6.6.2 demonstrated the design of an MDP that reasons about the addition or hiding of menu items, assuming that user characteristics are fully observable. While these approaches employ heuristic solution methods, they both have demonstrated positive value as reported in our experiments (both in simulation experiments and, the former, in a user study as well). Thus, better approximation of sequentially optimal

(POMDP) policy should be explored.

7.3 Evaluation Approaches

We found the combined use of simulation experiments and user studies to be an effective and informative way in evaluating system performance. In particular, the ability to use simulations to exhaustively check system performance under each parameter setting is extremely helpful.

In the development of predictive models, (objective) accuracy is often used to measure system performance. As mentioned above, we have used both objective and subjective utility as performance measures in our evaluations. We stress the importance of using subjective utility, due to the empirical evidence that people do not always prefer what is best objectively — what appears to be better for users subjectively may in fact require more effort without them realizing it.

In our user studies, we conducted within-subject experiments where each user evaluated four systems in one session that lasted between 60 to 90 minutes on average. Comparative evaluations often consist of fewer comparison systems, but we found that using four comparison systems worked, users did not find the experiments too tiring, and users were able to consistently distinguish the behaviour of each system. To facilitate the evaluation, we allowed users to take break and to write down their impressions of each system after each session. In one case, we also asked system-specific questions to help users remember their usage experience. We have not attempted a comparative evaluation with more than four systems, or conducted sessions lasting more than 90 minutes.

The importance of using *actual* interaction experiences to help users assess their preferences and determine how they differ from one’s conceptualized expectations was illustrated in Section 6.7. This finding is consistent with the literature in behavioural decision theory that suggests what people think they want is not always what they actu-

ally want. Therefore, methods that try to elicit user preferences upfront (in a conceptual manner) and do not attempt to re-elicite those preferences after some exposure to the software will likely fail to accommodate to individual users. Simple solutions to these approaches are to either change the type of elicitation questions so as to probe at experiential, rather than conceptual, preferences, or to ask users about their preferences only after some amount of software use.

7.4 Future Work

In the discussion above, we mentioned the need for future work in several cases. In particular, specific simulation and/or user experiments can be designed to directly address the following hypotheses:

- In a system that offers help for sequential goals (e.g., productivity software with suggestions for highlighting goals), the value of help as perceived by users is higher when the system uses an adaptive policy that can update its suggestions after each observation (even before a goal is completed) than one that cannot incrementally update its suggestions.

As a follow-up study to the user experiment presented in Chapter 5, this experiment would be a specific investigation designed to assess the value of incremental goal inference techniques in interface customization help systems.

- A system that uses probabilistic goal inference to offer help for sequential goals has higher perceived value than a system that uses non-probabilistic goal inference to offer help for those goals.

Similar to the previous point, the specific interest here is to directly compare the impact of help suggestions when the system uses probabilistic

goal inference to that when non-probabilistic goal inference is used. However, unlike the setup presented in Chapter 5, it would be worthwhile to explore this hypothesis in cases where the number of suggestions is small (e.g., $J = 3$) but the number of goals in the library are realistically large (e.g., $N = 50, 100, 500$). This extension would enable us to better assess the system's ability to select useful suggestions based on the information provided by the goal model.

- In a system that offers multiple help actions simultaneously to assist users carry out sequential goals, the perceived value of help is different when the system's action selection strategy chooses the *most likely* goals as suggestions rather than the *most useful* goals with respect to the diversity of suggestions made simultaneously (i.e., according to our definition of joint expected savings used in the case studies in Section 4.3, Section 4.4, and Chapter 5).

As a follow-up study to the pilot user experiment presented in Section 4.3 in the context of a word prediction help system that suggests three words at a time, this experiment would be a specific investigation designed to assess the value of an action selection strategy for joint actions.

- A help system that offers help using a long term policy has higher perceived value than one that offers help using a myopic policy.

While a long term policy is theoretically better than a myopic one, it would be interesting to explore if users perceive value in the same way. In particular, if such systems take exploratory actions, the experiment should measure the users' reaction to those system actions, and assess whether users perceive such systems to learn about them more quickly.

Moreover, we elaborate on some future work directions that require more in-depth explorations and development. These directions include the following topics:

- As mentioned at the beginning of this chapter, since empirical evidence suggests that goal inference accuracy need not be perfect in order for users to perceive value or utility in the help offered, an interesting study would be to identify the specific relationship between goal inference accuracy and overall system utility. For example, if goal inference accuracy can be controlled systematically and the overall system utility can be measured, we could determine when improvement in the goal model development affords the largest gain in system utility. Likewise, we could then determine the point of diminishing returns when additional development effort in the goal model is no longer worth the (potentially minimal) gain in perceived utility.
- In our case studies, we modeled a reduced number of interaction factors and assumed for simplicity that the utility function is additive. However, we saw in the experiential elicitation study that the utility function with the parameters neediness, length (number of suggestions), and quality was not additively decomposable. A future direction is to explore the use of more general utility functions in this domain. In addition, the utility structural findings from the experiential elicitation study showed that the structure varied most with the medium-level quality help (i.e., $Q = 2$) for varying values of length and neediness. This suggests that an intelligent software adaptation system is most uncertain about the user's preferences when the suggestions are partially correct. Thus, another future direction is to integrate active elicitation strategies in order to minimize this uncertainty early on.
- One of the implications of the DAISI framework is the ability for an intelligent system equipped with a User Characteristics Component to better adapt to different user preferences in comparison to the same system without a User Characteristics

Component. While we did not test this hypothesis directly, the initial results in the user study from Chapter 5 showed this was not the case. Specifically, we saw no significant differences between the policies GM (goal component only) and GC (goal and characteristics components). In some cases, GC did slightly worse than GM (e.g., average task completion times, percentage of acceptances, and several items on the post-questionnaire). Based on informal user feedback, we found that some users felt the GC policy was too sensitive and was unable to distinguish different reasons for rejecting help (as discussed earlier in this chapter). One possibility is to augment the observation model to take this information into consideration. Another possible investigation is to develop a richer User Characteristics Component that recognizes more than just the user's neediness so that a wider range of preferences can be accommodated.

- An in-depth exploration into learning user types and user characteristics, as well as their associated dynamics in the model is needed. The case studies throughout this thesis assumed a known model structure. An initial semi-supervised approach was used in Section 4.3 to collect data in an offline setting that investigates the number of hidden variables (user characteristics) in a confirmatory study. However, there was not enough data to acquire a definitive model and parameters. An obvious extension of this work is to develop more efficient ways for collecting (labeled or unlabeled) data involving user types and characteristics. Some data collection methods were reviewed in Chapter 2, but more work is needed to address the experiment needs involving hidden variables. Additional future work may involve incorporating techniques to learn DBNs with unknown structure and unknown parameters.

Other investigations include: automatically learning sequential goals; exploring whether users would like to explicitly control the system's level of interactivity in adaptive systems; integrating more personalization and online adaptation mechanisms; developing

techniques for eliciting subjective system utility throughout the interaction process; and integrating existing work that study interface transparency, trust between the system and user, and the utility of rationale.

Bibliography

- [AB04] P.D. Adamczyk and B.P. Bailey. If Not Now, When? The Effects of Interruption at Different Moments Within Task Execution. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pages 271–278, 2004.
- [AH02] Corin R. Anderson and Eric Horvitz. Web montage: A dynamic personalized start page. In *Proceedings of the World Wide Web Conference*, pages 7–11, 2002.
- [AZN98] D. Albrecht, I. Zukerman, and A. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction*, 8(1-2):5–47, 1998.
- [AZN99] D. Albrecht, I. Zukerman, and A. Nicholson. Pre-sending Documents on the WWW: A Comparative Study. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1274–1279, 1999.
- [BA04] Nate Blaylock and James Allen. Statistical goal parameter recognition. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 297–304, 2004.
- [Bai96] R.W. Bailey. *Human Performance Engineering: Designing High Quality Professional User Interfaces for Computer Products, Applications and Systems*. Prentice-Hall: Upper Saddle River, NJ, 1996.

- [BCM04] A. Bunt, C. Conati, and J. McGrenere. What role can adaptive support play in an adaptable system? In *Proceedings of Intelligent User Interfaces (IUI)*, pages 117–124, 2004.
- [BCM07] A. Bunt, C. Conati, and J. McGrenere. Supporting Interface Customization using a Mixed-Initiative Approach. In *Proceedings of Intelligent User Interfaces (IUI)*, pages 92–101, 2007.
- [BDH99] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.
- [Bel57] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.
- [BG95] Fahiem Bacchus and Adam Grove. Graphical models for preference and utility. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 3–10, Montreal, PQ, 1995.
- [BGBG95] R. Baecker, J. Grudin, W. Buxton, and S. Greenberg. *Readings in human-computer interaction: Towards the Year 2000*. Morgan Kaufmann, 1995.
- [BHD06] X. Bao, J. Herlocker, and T.G. Dietterich. Fewer clicks and less frustration: Reducing the cost of reaching the right folder. In *Proceedings of Intelligent User Interfaces (IUI)*, pages 178–185, 2006.
- [BJ01] Thorsten Bohnenberger and Anthony Jameson. When policies are better than plans: Decision-theoretic planning of recommendation sequences. In *Proceedings of Intelligent User Interfaces (IUI)*, pages 21–24, 2001.
- [BJJA05] Thorsten Bohnenberger, Oliver Jacobs, Anthony Jameson, and Ilhan Aslan. Decision-theoretic planning meets user requirements: Enhancements and

- studies of an intelligent shopping guide. In *Proceedings of the International Conference on Pervasive Computing*, pages 279–296, 2005.
- [BK98] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 33–42, Wisconsin, 1998.
- [BMC07] A. Bunt, J. McGrenere, and C. Conati. Understanding the Utility of Rationale in a Mixed-Initiative System for GUI Customization. In *Proceedings of User Modeling (UM)*, pages 147–156, 2007.
- [Bou02] Craig Boutilier. A POMDP formulation of preference elicitation problems. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)*, pages 239–246, 2002.
- [BPH⁺05] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1293–1299, 2005.
- [CCH01] E. Cutrell, M. Czerwinski, and E. Horvitz. Notification, Disruption, and Memory: Effects of Messaging Interruptions on Memory and Performance. In *Proc. of INTERACT*, pages 263–269, 2001.
- [CGG07] A. Cockburn, C. Gutwin, and S. Greenberg. A predictive model of menu performance. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, pages 627–636, 2007.
- [CGNS98] U. Chajewska, L. Getoor, J. Norman, and Y. Shahar. Utility elicitation as a classification problem. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 79–88, 1998.

- [CGV02] Cristina Conati, Abigail Gertner, and Kurt VanLehn. Using bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adaptive Interaction*, 12(4):371–417, 2002.
- [Che88] Hsien-Te Cheng. *Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, University of British Columbia, Vancouver, 1988.
- [CKO01] Urszula Chajewska, Daphne Koller, and Dirk Ormoneit. Learning an agent’s utility function by observing behavior. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 35–42, 2001.
- [CKP00] Urszula Chajewska, Daphne Koller, and Ronald Parr. Making rational decisions using adaptive utility elicitation. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)*, pages 363–369, 2000.
- [CLWB01] M. Claypool, P. Le, M. Waseda, and D. Brown. Implicit interest indicators. In *Proceedings of Intelligent User Interfaces (IUI)*, pages 33–40, 2001.
- [CM05] C. Conati and H. Maclaren. Data-driven refinement of a probabilistic model of user affect. In *Proceedings of User Modeling (UM)*, pages 40–49, 2005.
- [CM07] C. Conati and C. Merten. Eye-Tracking for User Modeling in Exploratory Learning Environments: an Empirical Evaluation. *Knowledge Based Systems*, 20(6):557–574, 2007.
- [CM09a] C. Conati and H. Maclaren. Empirically Building and Evaluating a Probabilistic Model of User Affect. *User Modeling and User-Adaptive Interaction*, 19(3):267–303, 2009.
- [CM09b] C. Conati and M. Manske. Evaluating Adaptive Feedback in an Educational Computer Game. In *Proceedings of International Conference on Intelligent Virtual Agents*, pages 146–158, 2009.

- [CMN80] S.K. Card, P.T. Moran, and A. Newell. *Psychology of human-computer interaction*. Hillsdale, NJ: Erlbaum, 1980.
- [CO87] J.M. Carroll and J.R. Olson. *Mental Models in Human-Computer Interaction*. National Academic Press, 1987.
- [Con02] C. Conati. Probabilistic assessment of user's emotions in educational games. *Journal of Applied Artificial Intelligence*, 16((7-8)):555–575, 2002.
- [Dec99] Rina Dechter. Bucket elimination: a unifying framework for reasoning. *Artificial Intelligence*, 113(1–2):41–85, 1999.
- [DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.
- [DLR77] A. Dempster, N. Laird, and D. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society, Series B*, 1(39):1–38, 1977.
- [Ebb85] H. Ebbinghaus. Memory: A Contribution to Experimental Psychology, 1885. Translated by H.A. Ruger and C.E. Bussenius. Retrieved on 01/08/2008 at <http://psychclassics.yorku.ca/Ebbinghaus/>.
- [FHA⁺05] J. Fogarty, S.E. Hudson, C.G. Atkeson, D. Avrahami, J. Forlizzi, S. Kiesler, J.C. Lee, and J. Yang. Predicting human interruptibility with sensors. *ACM Transactions on Computer-Human Interaction*, 12(1):119–146, March 2005.
- [Fis82] Peter C. Fishburn. *The Foundations of Expected Utility*. D. Reidel, Dordrecht, 1982.
- [Fit54] P.M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47:381–391, 1954.

- [FLL02a] G. Foster, P. Langlais, and G. Lapalme. Transtype: Text prediction for translators. In *Proc. of ACL Demonstrations*, pages 93–94, 2002.
- [FLL02b] G. Foster, P. Langlais, and G. Lapalme. User-friendly text prediction for translators. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2002.
- [GCH⁺05] K. Gajos, D. Christianson, R. Hoffmann, T. Shaked, K. Henning, J. Long, and D. Weld. Fast and robust interface generation for ubiquitous applications. In *Proceedings of the International Conference on Ubiquitous Computing (UBICOMP)*, pages 37–55, 2005.
- [GCTW06] K. Gajos, M. Czerwinski, D.S. Tan, and D.S. Weld. Exploring the Design Space For Adaptive Graphical User Interfaces. In *Proc. of AVI*, pages 201–208, 2006.
- [Gor83] R.L. Gorsuch. *Factor Analysis*. Hillsdale, NJ: Lawrence Erlbaum, 1983.
- [Gor99] Geoffrey J. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, Brown Carnegie Mellon university, Pittsburgh, 1999.
- [GP00] Peter Gorniak and David Poole. Building a stochastic dynamic model of application use. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 230–237, 2000.
- [GS83] D. Gentner and A.L. Stevens, editors. *Mental Models*. Lawrence Erlbaum Associates, 1983.
- [Gun68] R. Gunning. *The techniques of clear writing*. New York: McGraw-Hill, 1968.
- [GW88] S. Greenberg and I. Witten. Directing the user interface: how people use command-based computer systems. In *Proc. of IFAC Conference on Man-Machine Systems*, pages 349–356, 1988.

- [GW04] K.Z. Gajos and D.S. Weld. SUPPLE: automatically generating user interfaces. In *Proceedings of Intelligent User Interfaces (IUI)*, pages 93–100, 2004.
- [GW05] K.Z. Gajos and D. Weld. Preference elicitation for interface optimization. In *Proceedings of the Symposium on User Interface Software and Technology (UIST)*, pages 173–182, 2005.
- [HA03] Eric Horvitz and Johnson Apacible. Learning and reasoning about interruption. In *Proceedings of the International Conference on Multimodal Interfaces (ICMI)*, pages 20–27, Vancouver, Canada, 2003.
- [Han98] Eric A. Hansen. Solving POMDPs by searching in policy space. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 211–219, Madison, Wisconsin, 1998.
- [Han08] Eric A. Hansen. Sparse stochastic finite-state controllers for POMDPs. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 256–263, 2008.
- [HB06a] B. Hui and C. Boutilier. Modeling the Disruption to the User’s Mental Model. Neural Information Processing Systems (NIPS), Workshop on User Adaptive Systems, 2006.
- [HB06b] B. Hui and C. Boutilier. Who’s Asking for Help? A Bayesian Approach to Intelligent Assistance. In *Proceedings of Intelligent User Interfaces (IUI)*, pages 186–193, 2006.
- [HB08] B. Hui and C. Boutilier. Toward Eliciting Experienced Utilities for Interface Customization. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 298–305, 2008.

- [HBH⁺98] Eric Horvitz, Jack Breese, David Heckerman, David Hovel, and Koos Rommelse. The Lumière Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 256–265, 1998.
- [HD96] C. Huang and A. Darwiche. Inference in Belief Networks: A Procedural Guide. *International Journal of Approximate Reasoning*, 15(3):225–263, 1996.
- [HG00] T. Hasselbring and C. Glaser. Use of computer technology to help students with special needs. *The Future of Children: Children and Computer Technology*, 10(2):102–122, 2000.
- [HGB07] M. Hou, M.S. Gauthier, and S. Banbury. Development of a generic design framework for intelligent adaptive systems. In *Proceedings of Human-Computer Interaction International (HCI)*, volume 3, pages 313–320, 2007.
- [HGIB08] B. Hui, S. Gustafson, P. Irani, and C. Boutilier. The Need for an Interaction Cost Model. In *Proceedings of Advances in Visual Interfaces (AVI)*, pages 458–461, 2008.
- [Hic52] W. Hick. On the rate of gain of information. *Journal of Experimental Psychology*, 4:11–36, 1952.
- [HJH99] E. Horvitz, A. Jacobs, and D. Hovel. Attention-sensitive alerting. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 305–313, 1999.
- [HK97a] A. Hornof and D. Kieras. Cognitive modeling reveals menu search is both random and systematic. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, pages 107–114, 1997.

- [HK97b] A. Hornof and D. Kieras. Cognitive modeling reveals menu search is both random and systematic. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, pages 107–114, 1997.
- [HKA04] E. Horvitz, P. Koch, and J. Apacible. BusyBody: Creating and Fielding Personalized Models of the Cost of Interruption. In *Proceedings of Computer Supported Cooperative Work (CSCW)*, 2004.
- [HKLP07] Kaijen Hsiao, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Grasping POMDPs. In *IEEE Conference on Robotics and Automation*, 2007.
- [HKPH03] E. Horvitz, C.M. Kadie, T. Paek, and D. Hovel. Models of attention in computing and communications: From principles to applications. *Communications of the ACM*, 46(3):52–59, March 2003.
- [HKS⁺05] E. Horvitz, P. Koch, R. Sarin, J. Apacible, and M. Subramani. Bayesphone: Precomputation of context-sensitive policies for inquiry and action in mobile devices. In *Proceedings of User Modeling (UM)*, pages 251–260, 2005.
- [HLM03] B. Hui, S. Liaskos, and J. Mylopoulos. Requirements Analysis for Customizable Software: A Goals-Skills-Preferences Framework. In *Proceedings of Requirements Engineering (RE)*, pages 117–126, 2003.
- [Hor99a] Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, pages 159–166, 1999.
- [Hor99b] Eric Horvitz. Uncertainty, action, and interaction: In pursuit of mixed-initiative computing. *Intelligent Systems*, pages 17–20, 1999.
- [HP00] Eric Horvitz and Tim Paek. Deeplistener: Harnessing expected utility to guide clarification dialog in spoken language systems. In *Proceedings*

- of the International Conference on Spoken Language Processing (INTER-SPEECH)*, pages 226–229, 2000.
- [HPB09] B. Hui, G. Partridge, and C. Boutilier. A Probabilistic Mental Model For Estimating Disruption. In *Proceedings of Intelligent User Interfaces (IUI)*, pages 287–296, 2009.
- [HSAHB99] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. Spudd: Stochastic planning using decision diagrams. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 279–288, 1999.
- [Hym53] R. Hyman. Stimulus information as a determinant of reaction time. *Journal of Experimental Psychology*, 45:188–196, 1953.
- [HZ04] C.K. Hsee and J. Zhang. Distinction bias: Misprediction and mischoice due to joint evaluation. *Personality and Social Psychology*, 86(5):680–695, 2004.
- [Jam01] Anthony Jameson. Modeling both the context and the user. *Personal and Ubiquitous Computing*, 5(1):29–33, 2001.
- [JGHM⁺01] Anthony Jameson, Barbara Großmann-Hutter, Leonie March, Ralf Rummer, Thorsten Bohnenberger, and Frank Wittig. When actions have consequences: Empirically based decision making for intelligent user interfaces. *Knowledge-Based Systems*, 14:75–92, 2001.
- [JKM⁺09] Anthony Jameson, Juergen Kiefer, Christian Müller, Barbara Großmann-Hutter, Frank Wittig, and Ralf Rummer. Assessment of a users time pressure and cognitive load on the basis of features of speech. In Matthew Crocker and Joerg Siekmann, editors, *Resource-Adaptive Cognitive Processes*. Springer, Berlin, 2009.

- [JL83] P.N. Johnson-Laird. *Mental Models: Toward a Cognitive Science of Language, Inference and Consciousness*. Harvard University Press, 1983.
- [KBP07] A. Kapoor, W. Bursleson, and R.W. Picard. Automatic prediction of frustration. *International Journal of Human-Computer Studies*, 65(8):724–736, 2007.
- [KH05] A. Kapoor and E. Horvitz. Principles of lifelong learning for predictive user modeling. In *Proceedings of User Modeling (UM)*, pages 37–46, 2005.
- [KKL02] J. Kay, B. Kummerfeld, and P. Lauder. Personis: A server for user modeling. In *Proceedings of International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH)*, pages 201–212, 2002.
- [KKLL99] K. Kang, S. Kim, J.J. Lee, and K.W. Lee. Feature-oriented engineering of pbx software for adaptability and reuseability. *Software Practice and Experience*, 29(10):875–896, 1999.
- [KL51] S. Kullback and R. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.
- [Kob95] Alfred Kobsa. Supporting user interfaces for all through user modeling. In *Proceedings of Human-Computer Interaction International (HCI)*, pages 155–157, 1995.
- [Kob01] Alfred Kobsa. Generic user modeling systems. *User Modeling and User-Adaptive Interaction*, 11(1-2):49–63, 2001.
- [KR76] R.L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*. Wiley, NY, 1976.

- [KS90] D. Kahneman and J. Snell. Predicting utility. In R.M. Hogarth, editor, *Insights in Decision Making*, pages 295–310. University of Chicago Press, 1990.
- [KT06] D. Kahneman and R.H. Thaler. Utility maximization and experienced utility. *Economic Perspectives*, 20(1):221–234, 2006.
- [KWS97] D. Kahneman, P.P. Wakker, and R. Sarin. Back to bentham? explorations of experienced utility. *Quarterly Journal of Economics*, 112(2):375–405, 1997.
- [Les97] N. Lesh. Adaptive goal recognition. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1208–1214, 1997.
- [LHL97] G. Linden, S. Hanks, and N. Lesh. Interactive assessment of user preference models: The automated travel assistant. In *Proceedings of User Modeling (UM)*, pages 67–78, 1997.
- [LS88] S. Lauritzen and D. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society Series B*, 50:157–224, 1988.
- [LZZ⁺06] Wenhui Liao, Weihong Zhang, Zhiwei Zhu, Qiang Ji, and Wayne Gray. Toward a Decision-Theoretic Framework for Affect Recognition and User Assistance. *International Journal of Human-Computer Studies*, 64(9):847–873, 2006.
- [Mac95] I.S. MacKenzie. Movement time prediction in human-computer interfaces. In R.M. Baecker, W.A.S. Buxton, J. Grudin, and S. Greenberg, editors, *Readings in human-computer interaction (2nd ed.)*, pages 483–493. Los Altos, CA: Kaufmann, 1995.

- [MBB02] 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 2002, ACM CHI Letters 4(1)*, pages 163–170, 2002.
- [MC06] D. Metzler and W.B. Croft. Beyond bags of words: Modeling implicit user preferences in information retrieval. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)*, 2006.
- [McG00] J. McGrenere. Bloat: The objective and subject dimension. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pages 337–338, 2000.
- [McG02] J. McGrenere. *The design and evaluation of multiple interfaces: A solution for complex software*. PhD thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 2002.
- [MHC99] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, pages 541–548, Orlando, FL, 1999.
- [Mic] <http://office.microsoft.com/>.
- [MM01] Michael Mayo and Antonija Mitrovic. Optimising ITS Behaviour with Bayesian Networks and Decision Theory. *International Journal of Artificial Intelligence in Education*, 12:124–153, 2001.
- [Mon82] George E. Monahan. A survey of partially observable Markov decision processes: Theory, models and algorithms. *Management Science*, 28:1–16, 1982.
- [MPKK99] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments.

- In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 427–436, Stockholm, 1999.
- [Mur02] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, Department of Computer Science, UC Berkeley, CA, USA, 2002.
- [MV00] C. Murray and K. VanLehn. DT Tutor: A decision-theoretic dynamic approach for optimal selection of tutorial actions. In *Proceedings of the International Conference on Intelligent Tutoring Systems (ITS)*, pages 153–162, Montréal, Canada, 2000. Springer.
- [Nor83] D.A. Norman. Some Observations on Mental Models. In D. Gentner and A.L. Stevens, editors, *Mental Models*. Lawrence Erlbaum Associates, 1983.
- [Nor94] Donald A. Norman. How might people interact with agents? *Communications of ACM*, 37(7):68–71, 1994.
- [PB03] P. Poupart and C. Boutilier. Value-directed Compression of POMDPs. In *Proceedings of Conference on Neural Information Processing Systems (NIPS)*, pages 1547–1554, 2003.
- [PB04] Pascal Poupart and Craig Boutilier. Bounded Finite State Controllers. In *Proceedings of Conference on Neural Information Processing Systems (NIPS)*, 2004.
- [Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [PFT03] P. Pu, B. Faltings, and M. Torrens. User-involved preference elicitation. In *IJCAI Workshop on Configuration*, 2003.

- [PGT03] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1025–1032, 2003.
- [PMI05] Helmut Prendinger, Junichiro Mori, and Mitsuru Ishizuka. Recognizing, modeling, and responding to users’ affective states. In *Proceedings of User Modeling (UM)*, pages 60–69, 2005.
- [PMP⁺03] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun. Towards robotic assistants in nursing homes: Challenges and results. *Special issue on Socially Interactive Robots, Robotics and Autonomous Systems*, 42(3-4):271–281, 2003.
- [Put94] M. L. Puterman. *Markov Decision Problems*. Wiley, New York, 1994.
- [RG03] Nicholas Roy and Geoffrey Gordon. Exponential Family PCA for Belief Compression in POMDPs. In *Proceedings of Conference on Neural Information Processing Systems (NIPS)*, pages 1043–1049, 2003.
- [RIS⁺94] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work (CSCW)*, pages 175–186, 1994.
- [RLH75] L. Ross, M.R. Lepper, and M. Hubbard. Perversance in self-perception and social perception: Biased attributional processes in the debriefing paradigm. *Journal of Personality and Social Psychology*, 32:880–892, 1975.
- [RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

- [RW92] A. Rutherford and J.R. Wilson. Searching for mental models in human-machine systems. In *Models in the Mind*, pages 195–224. Academic Press Inc., 1992.
- [SAHB00] R. St-Aubin, J. Hoey, and C. Boutilier. Apricodd: Approximate policy construction using decision diagrams. In *Proceedings of Neural Information Processing Systems (NIPS) Conference*, pages 1089–1095, 2000.
- [Sas97] M.A. Sasse. *Eliciting and Describing User’s Models of Computer Systems*. PhD thesis, School of Computer Science, The University of Birmingham, Birmingham, England, 1997.
- [SB10] Ross D. Shacter and Debarun Bhattacharjya. Solving influence diagrams: Exact algorithms. In *Wiley Encyclopedia of Operations Research and Management Science*. Wiley, 2010.
- [SBD⁺05] S. Stumpf, X. Bao, A. Dragunov, T.G. Dietterich, J. Herlocker, K. Johnsrude, L. Li, and J.Q. Shen. The tasktracer system. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1712–1713, 2005.
- [SBH02] Guy Shani, Ronen I. Brafman, and David Heckerman. An MDP-Based Recommender System. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 453–460, 2002.
- [SJ04] Holger Schultheis and Anthony Jameson. Assessing cognitive load in adaptive hypermedia systems: Physiological and behavioral methods. In Wolfgang Nejdl and Paul De Bra, editors, *Proceedings of International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH)*, pages 225–234. Springer, Berlin, 2004.

- [SL01] S. Shearin and H. Lieberman. Intelligent profiling by example. In *Proceedings of Intelligent User Interfaces (IUI)*, pages 145–151, 2001.
- [SM97] Ben Shneiderman and Pattie Maes. Direct manipulation vs. interface agents: Excerpts from debates at iui 97 and chi 97. *ACM Interactions*, 4(6):42–46, 1997.
- [SNN⁺01] F. Shein, T. Nantais, R. Nishiyama, C. Tam, and P. Marshall. Word cueing for persons with writing difficulties: WordQ. In *Proceedings of the Annual International Conference on Technology and Persons with Disabilities*, Los Angeles, CA, USA, 2001.
- [Son71] Edward J. Sondik. *The optimal control of partially observable Markov Decision Processes*. PhD thesis, Stanford university, Palo Alto, 1971.
- [SS94] A. Sears and B. Schneiderman. Split menus: Effectively using selection frequency to organize menus. *ACM Transactions on Computer-Human Interaction*, 1(1):27–51, 1994.
- [STZ05] Xuehua Shen, Bin Tan, and Cheng Xiang Zhai. Implicit user modeling for personalized search. In *Proceedings of the International Conference on Information Knowledge Management*, pages 824–831, 2005.
- [SV04] Matthijs T.J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for POMDPs. Technical Report IAS-UVA-04-02, Informatics Institute, University of Amsterdam, November 2004.
- [TDCF07] Joe Tullio, Anind Dey, Jason Chalecki, and James Fogarty. How it works: A field study of non-technical users interacting with an intelligent system. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pages 31–40, 2007.

- [Tms05] T. Tsandilas and m.c. schraefel. An empirical assessment of adaptation techniques. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI): Extended Abstracts*, pages 2009–2012, 2005.
- [Vas04] Julita Vassileva. A practical architecture for user modeling in a hypermedia-based information system. In *Proceedings of User Modeling (UM)*, pages 115–120, 2004.
- [War01] J. Warren. Cost/Benefit Based Adaptive Dialog: Case Study Using Empirical Medical Practice Norms and Intelligent Split Menus. In *Proc. of AUIC*, pages 100–107, 2001.
- [WY05] Jason D. Williams and Steve Young. Scaling Up POMDPs for Dialog Management: The “Summary POMDP” Method. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 177–182, 2005.
- [ZC03] Xiaoming Zhou and Cristina Conati. Inferring user goals from personality and behavior in a causal model of user affect. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*, pages 211–218, 2003.
- [Zip49] G.K. Zipf. *Human Behavior and the Principle of Least-Effort*. Addison-Wesley, 1949.

Appendix A

Materials

In this appendix, we provide the questionnaire used in the factor analysis in Chapter 4.

A.1 Chapter 4 Questionnaire for User Types

The statements below are about your attitudes toward the computer behaviour that you have just experienced. Using the scale below as a guide, write a number on the line in front of each statement to indicate the extent to which you agree or disagree with each statement. Be honest and as accurate as possible. Use the following scale:

1	—	2	—	3	—	4	—	5
Strongly				Neutral				Strongly
Disagree								Agree

- 1. I preferred typing by myself without help.
- 2. I found the flashing menu items distracting.
- 3. I did not become more frustrated when the system lagged.
- 4. I accepted suggestions often.
- 5. I found reading through the words in the suggestion distracting.
- 6. I would not have been as frustrated if the suggestions appeared elsewhere on the screen.
- 7. I accepted suggestions that were not the correct words.
- 8. I was not interested in exploring other system functionality.
- 9. Background noises did not aggravate me.
- 10. I learned to type the new keyboard layout quickly.
- 11. The background noises did not prevent me from concentrating on the task.
- 12. I did not like the general computing environment.
- 13. I did not value the suggestions.
- 14. The animation did not catch my attention during the task.
- 15. Not having the correct words in the suggestions made me annoyed.
- 16. I think almost everyone could benefit from accepting the suggestions.
- 17. I spent a great deal of time closing the pop-up boxes.
- 18. The suggestions made the process less frustrating.

Appendix B

Parameter Specification

In this appendix, we document the detailed parameters used in the experiments presented in the thesis.

B.1 Chapter 4 Case Study One Parameters

The parameters used in the DBN in Section 4.3 are defined as follows. We first define the CPTs in the first time slice of the DBN:

Pr(TF)					Pr(TN)					
no	yes				no	yes				
.5	.5				.5	.5				
Pr(TI)					Pr(TD)					
low	med	high				low	med	high		
.33	.34	.33				.33	.34	.33		
Pr(F _t TF)					Pr(N _t TN)					
TF	low	med	high				TN	low	med	high
no	.7	.2	.1				no	.7	.2	.1
yes	.1	.2	.7				yes	.1	.2	.7
Pr(Cons _t BOX _t ,TD,TI,N _t ,F _t)										
BOX _t	TD	TI	N _t	F _t	low	med	high			
off	*	*	*	*	1.0	0.0	0.0			
on	*	*	*	*	.33	.34	.33			

The observation model $Pr(Obs_t|Q_t, S_t, TD, TI, N_t, F_t, Cons_t)$ is common to both time slices. There are 22 observations in total: continuous key down (minor), continuous key down (major), mouse back/forth (minor), mouse back/forth (major), jam into keyboard (minor), jam into keyboard (major), continuous mouse clicks (minor), continuous mouse clicks (major), erasing many chars (minor), erasing many chars (major), surfing menus (minor), surfing menus (major), switching applications (minor), switching applications (major), pause, slider up, slider down, type, accept help, hover help (minor), hover help (major), and help pause. For space reasons, we do not enumerate the parameters of this function. Rather, we provide the Matlab code used to generate it.

```
function cpt = genObsmat
global QUALsz SDRsz CHARsz OBSsz accx;
cpt = zeros(QUALsz,SDRsz,CHARsz,CHARsz,CHARsz,CHARsz,CHARsz,OBSsz);
%           Q     S     D     I     N     F     C     O
FEWW = 1;
SOME = 3;
LOTS = 5;
lo = 1;
me = 2;
hi = 3;
f2dices = [vocabdex('F1a') vocabdex('F2a') vocabdex('F3a') vocabdex('F4a')];
f3dices = [vocabdex('F1b') vocabdex('F2b') vocabdex('F3b') vocabdex('F4b')];
n1ax = [vocabdex('N1a')];
n1bx = [vocabdex('N1b')];
nd2dices = [vocabdex('ND1a') vocabdex('ND2a') ];
nd3dices = [vocabdex('ND1b') vocabdex('ND2b') ];
pawx = [vocabdex('PAUSE')];
supx = vocabdex( 'SDRup' );
sdnx = vocabdex( 'SDRdn' );
typex = [vocabdex('TYPE')];
consdices = [vocabdex('HH2') vocabdex('HH3') vocabdex('HP')];
for Q=1:QUALsz, for S=1:SDRsz,
for F=1:CHARsz, for N=1:CHARsz, for D=1:CHARsz, for I=1:CHARsz,
for C=1:CHARsz,
ovec = .001*ones(1,OBSsz);
%%% define behavioural pattern
if F == lo, ovec( typex ) = LOTS*2;
```

```

elseif F == me, ovec( f2dices ) = LOTS;
                ovec( typex )   = FEWW;
elseif F == hi, ovec( f3dices ) = LOTS*2;
                ovec( typex )   = FEWW;

end;

if    N == lo, ovec( typex )     = LOTS*2;
                ovec( sdnx )     = FEWW;
elseif N == me, ovec( n1ax )    = FEWW;
                ovec( nd2dices ) = FEWW;
                ovec( typex )    = SOME;
elseif N == hi, ovec( n1bx )    = LOTS;
                ovec( pawx )     = FEWW;
                ovec( typex )    = FEWW;

end;

if    D == lo, ovec( typex )     = LOTS*2;
elseif D == me, ovec( nd2dices ) = FEWW;
                ovec( pawx )     = SOME;
                ovec( typex )    = LOTS;
elseif D == hi, ovec( nd3dices ) = FEWW;
                ovec( pawx )     = LOTS;
                ovec( typex )    = SOME;

end;

if    I == lo, ovec( supx )      = ovec( supx ) + FEWW;
                ovec( typex )    = LOTS-1;
elseif I == me, ovec( sdnx )    = FEWW;
                ovec( typex )    = LOTS;
elseif I == hi,
                ovec( typex )    = LOTS*2;

end;

%% personality and suggestion considerations:
if Q <= 4,
    if I == hi, ovec( accx )     = FEWW;
                ovec( consdices ) = FEWW;

    end;

end;

if Q <= 2,
    if N == hi, ovec( accx )     = LOTS;
                ovec( consdices ) = LOTS;

    end;

if I == hi, ovec( accx )     = LOTS;

end;

```

```

end;
% amend troublesome conditions
if S == 1, ovec( sdnx ) = FEWW; end;
if S == 5, ovec( supx ) = FEWW; end;
%%% consideration case CONS=1:
if C == lo,
    ovec( accx:length(ovec) ) = 0;
%%% consideration case CONS=2:
elseif C == me,
    if Q >= 2, ovec( consdices ) = LOTS;
    else      ovec( consdices ) = FEWW;
    end;
    if      Q >= 4, ovec( accx ) = LOTS;
    elseif Q >= 2, ovec( accx ) = LOTS;
    end;
    cpt(Q,S,D,I,N,F,C,:) = normalize( ovec );
%%% consideration case CONS=3:
elseif C == hi,
    if Q >= 2, ovec( consdices ) = LOTS;
    else      ovec( consdices ) = FEWW;
    end;
    if      Q >= 4, ovec( accx ) = LOTS;
    elseif Q >= 2, ovec( accx ) = LOTS;
    end;
end;
% add vector
cpt(Q,S,D,I,N,F,C,:) = normalize( ovec );
end; end; end; end; end;
end; end;

```

The transition distributions in the DBN are:

Pr(F _t TF,F _{t-1})				
TF	F _{t-1}	low	med	high
no	low	.7	.2	.1
no	med	.3	.5	.2
no	high	.1	.2	.7
yes	low	.4	.4	.2
yes	med	.2	.5	.3
yes	high	.1	.1	.8

Pr(N_t|TN,N_{t-1})

TN	N _{t-1}	low	med	high
no	low	.7	.2	.1
no	med	.3	.5	.2
no	high	.1	.2	.7
yes	low	.4	.4	.2
yes	med	.2	.5	.3
yes	high	.1	.1	.8

						Pr(Cons _t Cons _{t-1} ,BOX _t ,TD,TI,N _t ,F _t)		
Cons _t	BOX _t	TD	TI	N _t	F _t	low	med	high
*	off	*	*	*	*	1.0	0.0	0.0
low	on	low	low	low	low	.20	.30	.50
med	on	low	low	low	low	.10	.30	.60
high	on	low	low	low	low	.10	.30	.60
low	on	low	low	low	med	.20	.35	.45
med	on	low	low	low	med	.10	.35	.55
high	on	low	low	low	med	.10	.35	.55
low	on	low	low	low	high	.30	.30	.40
med	on	low	low	low	high	.20	.30	.50
high	on	low	low	low	high	.10	.30	.60
low	on	low	low	med	low	.20	.20	.60
med	on	low	low	med	low	.10	.20	.70
high	on	low	low	med	low	.10	.20	.70
low	on	low	low	med	med	.20	.25	.55
med	on	low	low	med	med	.10	.25	.65
high	on	low	low	med	med	.10	.25	.65
low	on	low	low	med	high	.20	.30	.50
med	on	low	low	med	high	.10	.30	.60
high	on	low	low	med	high	.10	.30	.60
low	on	low	low	high	low	.11	.09	.80
med	on	low	low	high	low	.01	.09	.90
high	on	low	low	high	low	.01	.09	.90
low	on	low	low	high	med	.15	.10	.75
med	on	low	low	high	med	.05	.10	.85
high	on	low	low	high	med	.05	.10	.85
low	on	low	low	high	high	.20	.10	.70
med	on	low	low	high	high	.10	.10	.80
high	on	low	low	high	high	.10	.10	.80
low	on	low	med	low	low	.25	.30	.45
med	on	low	med	low	low	.15	.30	.55

high	on	low	med	low	low	.05	.30	.65
low	on	low	med	low	med	.25	.35	.40
med	on	low	med	low	med	.15	.35	.50
high	on	low	med	low	med	.05	.35	.60
low	on	low	med	low	high	.35	.30	.35
med	on	low	med	low	high	.25	.30	.45
high	on	low	med	low	high	.15	.30	.55
low	on	low	med	med	low	.25	.20	.55
med	on	low	med	med	low	.15	.20	.65
high	on	low	med	med	low	.05	.20	.75
low	on	low	med	med	med	.25	.25	.50
med	on	low	med	med	med	.15	.25	.60
high	on	low	med	med	med	.05	.25	.70
low	on	low	med	med	high	.25	.30	.45
med	on	low	med	med	high	.15	.30	.55
high	on	low	med	med	high	.05	.30	.65
low	on	low	med	high	low	.16	.09	.75
med	on	low	med	high	low	.06	.09	.85
high	on	low	med	high	low	.06	.09	.85
low	on	low	med	high	med	.20	.10	.70
med	on	low	med	high	med	.10	.10	.80
high	on	low	med	high	med	.10	.10	.80
low	on	low	med	high	high	.25	.10	.65
med	on	low	med	high	high	.15	.10	.75
high	on	low	med	high	high	.05	.10	.85
low	on	low	high	low	low	.30	.35	.35
med	on	low	high	low	low	.20	.35	.45
high	on	low	high	low	low	.10	.35	.55
low	on	low	high	low	med	.30	.40	.30
med	on	low	high	low	med	.20	.40	.40
high	on	low	high	low	med	.10	.40	.50
low	on	low	high	low	high	.40	.35	.25
med	on	low	high	low	high	.30	.35	.35
high	on	low	high	low	high	.20	.35	.45
low	on	low	high	med	low	.30	.25	.45
med	on	low	high	med	low	.20	.25	.55
high	on	low	high	med	low	.10	.25	.65
low	on	low	high	med	med	.30	.30	.40
med	on	low	high	med	med	.20	.30	.50
high	on	low	high	med	med	.10	.30	.60

low	on	low	high	med	high	.30	.35	.35
med	on	low	high	med	high	.20	.35	.45
high	on	low	high	med	high	.10	.35	.55
low	on	low	high	high	low	.21	.14	.65
med	on	low	high	high	low	.11	.14	.75
high	on	low	high	high	low	.01	.14	.85
low	on	low	high	high	med	.25	.15	.60
med	on	low	high	high	med	.15	.15	.70
high	on	low	high	high	med	.05	.15	.80
low	on	low	high	high	high	.30	.15	.55
med	on	low	high	high	high	.20	.15	.65
high	on	low	high	high	high	.10	.15	.75
low	on	med	low	low	low	.30	.40	.30
med	on	med	low	low	low	.20	.40	.40
high	on	med	low	low	low	.10	.40	.50
low	on	med	low	low	med	.30	.45	.25
med	on	med	low	low	med	.20	.45	.35
high	on	med	low	low	med	.10	.45	.45
low	on	med	low	low	high	.30	.50	.20
med	on	med	low	low	high	.20	.50	.30
high	on	med	low	low	high	.10	.50	.40
low	on	med	low	med	low	.25	.35	.40
med	on	med	low	med	low	.15	.35	.50
high	on	med	low	med	low	.05	.35	.60
low	on	med	low	med	med	.25	.40	.35
med	on	med	low	med	med	.15	.40	.45
high	on	med	low	med	med	.05	.40	.55
low	on	med	low	med	high	.25	.45	.30
med	on	med	low	med	high	.15	.45	.40
high	on	med	low	med	high	.05	.45	.50
low	on	med	low	high	low	.20	.20	.60
med	on	med	low	high	low	.10	.20	.70
high	on	med	low	high	low	.10	.20	.70
low	on	med	low	high	med	.20	.25	.55
med	on	med	low	high	med	.10	.25	.65
high	on	med	low	high	med	.10	.25	.65
low	on	med	low	high	high	.20	.30	.50
med	on	med	low	high	high	.10	.30	.60
high	on	med	low	high	high	.10	.30	.60

low	on	med	med	low	low	0.3500	0.4000	0.2500
med	on	med	med	low	low	0.2500	0.4000	0.3500
high	on	med	med	low	low	0.1500	0.4000	0.4500
low	on	med	med	low	med	0.3500	0.4500	0.2000
med	on	med	med	low	med	0.2500	0.4500	0.3000
high	on	med	med	low	med	0.1500	0.4500	0.4000
low	on	med	med	low	high	0.3500	0.5000	0.1500
med	on	med	med	low	high	0.2500	0.5000	0.2500
high	on	med	med	low	high	0.1500	0.5000	0.3500
low	on	med	med	med	low	0.3000	0.3500	0.3500
med	on	med	med	med	low	0.2000	0.3500	0.4500
high	on	med	med	med	low	0.1000	0.3500	0.5500
low	on	med	med	med	med	0.3000	0.4000	0.3000
med	on	med	med	med	med	0.2000	0.4000	0.4000
high	on	med	med	med	med	0.1000	0.4000	0.5000
low	on	med	med	med	high	0.3000	0.4500	0.2500
med	on	med	med	med	high	0.2000	0.4500	0.3500
high	on	med	med	med	high	0.1000	0.4500	0.4500
low	on	med	med	high	low	0.2500	0.2000	0.5500
med	on	med	med	high	low	0.1500	0.2000	0.6500
high	on	med	med	high	low	0.0500	0.2000	0.7500
low	on	med	med	high	med	0.2500	0.2500	0.5000
med	on	med	med	high	med	0.1500	0.2500	0.6000
high	on	med	med	high	med	0.0500	0.2500	0.7000
low	on	med	med	high	high	0.2500	0.3000	0.4500
med	on	med	med	high	high	0.1500	0.3000	0.5500
high	on	med	med	high	high	0.0500	0.3000	0.6500
low	on	med	high	low	low	0.4000	0.4500	0.1500
med	on	med	high	low	low	0.3000	0.4500	0.2500
high	on	med	high	low	low	0.2000	0.4500	0.3500
low	on	med	high	low	med	0.4000	0.5000	0.1000
med	on	med	high	low	med	0.3000	0.5000	0.2000
high	on	med	high	low	med	0.2000	0.5000	0.3000
low	on	med	high	low	high	0.4000	0.5500	0.0500
med	on	med	high	low	high	0.3000	0.5500	0.1500
high	on	med	high	low	high	0.2000	0.5500	0.2500
low	on	med	high	med	low	0.3500	0.4000	0.2500
med	on	med	high	med	low	0.2500	0.4000	0.3500
high	on	med	high	med	low	0.1500	0.4000	0.4500
low	on	med	high	med	med	0.3500	0.4500	0.2000

med	on	med	high	med	med	0.2500	0.4500	0.3000
high	on	med	high	med	med	0.1500	0.4500	0.4000
low	on	med	high	med	high	0.3500	0.5000	0.1500
med	on	med	high	med	high	0.2500	0.5000	0.2500
high	on	med	high	med	high	0.1500	0.5000	0.3500
low	on	med	high	high	low	0.3000	0.2500	0.4500
med	on	med	high	high	low	0.2000	0.2500	0.5500
high	on	med	high	high	low	0.1000	0.2500	0.6500
low	on	med	high	high	med	0.3000	0.3000	0.4000
med	on	med	high	high	med	0.2000	0.3000	0.5000
high	on	med	high	high	med	0.1000	0.3000	0.6000
low	on	med	high	high	high	0.3000	0.3500	0.3500
med	on	med	high	high	high	0.2000	0.3500	0.4500
high	on	med	high	high	high	0.1000	0.3500	0.5500
low	on	high	low	low	low	0.4000	0.4000	0.2000
med	on	high	low	low	low	0.3000	0.4000	0.3000
high	on	high	low	low	low	0.2000	0.4000	0.4000
low	on	high	low	low	med	0.4000	0.4500	0.1500
med	on	high	low	low	med	0.3000	0.4500	0.2500
high	on	high	low	low	med	0.2000	0.4500	0.3500
low	on	high	low	low	high	0.4000	0.5000	0.1000
med	on	high	low	low	high	0.3000	0.5000	0.2000
high	on	high	low	low	high	0.2000	0.5000	0.3000
low	on	high	low	med	low	0.3000	0.4000	0.3000
med	on	high	low	med	low	0.2000	0.4000	0.4000
high	on	high	low	med	low	0.1000	0.4000	0.5000
low	on	high	low	med	med	0.3000	0.4500	0.2500
med	on	high	low	med	med	0.2000	0.4500	0.3500
high	on	high	low	med	med	0.1000	0.4500	0.4500
low	on	high	low	med	high	0.3000	0.5000	0.2000
med	on	high	low	med	high	0.2000	0.5000	0.3000
high	on	high	low	med	high	0.1000	0.5000	0.4000
low	on	high	low	high	low	0.2000	0.3000	0.5000
med	on	high	low	high	low	0.1000	0.3000	0.6000
high	on	high	low	high	low	0.1000	0.3000	0.6000
low	on	high	low	high	med	0.2000	0.3500	0.4500
med	on	high	low	high	med	0.1000	0.3500	0.5500
high	on	high	low	high	med	0.1000	0.3500	0.5500
low	on	high	low	high	high	0.2000	0.4000	0.4000
med	on	high	low	high	high	0.1000	0.4000	0.5000

high	on	high	low	high	high	0.1000	0.4000	0.5000
low	on	high	med	low	low	0.4500	0.4000	0.1500
med	on	high	med	low	low	0.3500	0.4000	0.2500
high	on	high	med	low	low	0.2500	0.4000	0.3500
low	on	high	med	low	med	0.4500	0.4500	0.1000
med	on	high	med	low	med	0.3500	0.4500	0.2000
high	on	high	med	low	med	0.2500	0.4500	0.3000
low	on	high	med	low	high	0.4500	0.5000	0.0500
med	on	high	med	low	high	0.3500	0.5000	0.1500
high	on	high	med	low	high	0.2500	0.5000	0.2500
low	on	high	med	med	low	0.3500	0.4000	0.2500
med	on	high	med	med	low	0.2500	0.4000	0.3500
high	on	high	med	med	low	0.1500	0.4000	0.4500
low	on	high	med	med	med	0.3500	0.4500	0.2000
med	on	high	med	med	med	0.2500	0.4500	0.3000
high	on	high	med	med	med	0.1500	0.4500	0.4000
low	on	high	med	med	high	0.3500	0.5000	0.1500
med	on	high	med	med	high	0.2500	0.5000	0.2500
high	on	high	med	med	high	0.1500	0.5000	0.3500
low	on	high	med	high	low	0.2500	0.3000	0.4500
med	on	high	med	high	low	0.1500	0.3000	0.5500
high	on	high	med	high	low	0.0500	0.3000	0.6500
low	on	high	med	high	med	0.2500	0.3500	0.4000
med	on	high	med	high	med	0.1500	0.3500	0.5000
high	on	high	med	high	med	0.0500	0.3500	0.6000
low	on	high	med	high	high	0.2500	0.4000	0.3500
med	on	high	med	high	high	0.1500	0.4000	0.4500
high	on	high	med	high	high	0.0500	0.4000	0.5500
low	on	high	high	low	low	0.5000	0.4500	0.0500
med	on	high	high	low	low	0.4000	0.4500	0.1500
high	on	high	high	low	low	0.3000	0.4500	0.2500
low	on	high	high	low	med	0.4000	0.5000	0.1000
med	on	high	high	low	med	0.4000	0.5000	0.1000
high	on	high	high	low	med	0.3000	0.5000	0.2000
low	on	high	high	low	high	0.4000	0.5500	0.0500
med	on	high	high	low	high	0.4000	0.5500	0.0500
high	on	high	high	low	high	0.3000	0.5500	0.1500
low	on	high	high	med	low	0.4000	0.4500	0.1500
med	on	high	high	med	low	0.3000	0.4500	0.2500

high	on	high	high	med	low	0.2000	0.4500	0.3500
low	on	high	high	med	med	0.4000	0.5000	0.1000
med	on	high	high	med	med	0.3000	0.5000	0.2000
high	on	high	high	med	med	0.2000	0.5000	0.3000
low	on	high	high	med	high	0.4000	0.5500	0.0500
med	on	high	high	med	high	0.3000	0.5500	0.1500
high	on	high	high	med	high	0.2000	0.5500	0.2500
low	on	high	high	high	low	0.3000	0.3500	0.3500
med	on	high	high	high	low	0.2000	0.3500	0.4500
high	on	high	high	high	low	0.1000	0.3500	0.5500
low	on	high	high	high	med	0.3000	0.4000	0.3000
med	on	high	high	high	med	0.2000	0.4000	0.4000
high	on	high	high	high	med	0.1000	0.4000	0.5000
low	on	high	high	high	high	0.3000	0.4500	0.2500
med	on	high	high	high	high	0.2000	0.4500	0.3500
high	on	high	high	high	high	0.1000	0.4500	0.4500

The reward and cost models are defined as follows:

TD	TI	N _t	Q _{t=1}			Q _{t=2}			Q _{t=3}			Q _{t=4,5}		
			F _t			F _t			F _t			F _t		
			low	med	high	low	med	high	low	med	high	low	med	high
low	low	low	2	2	2	30	27	25	41	40	39	44	44	44
low	low	med	5	5	5	31	28	26	41	40	39	44	44	44
low	low	high	9	9	9	32	29	27	41	40	39	44	44	44
low	med	low	2	2	2	31	28	26	39	38	37	42	42	42
low	med	med	3	3	3	31.5	28.5	26.5	40	39	38	43	43	43
low	med	high	5	5	5	32	29	27	41	40	39	44	44	44
low	high	low	1	1	1	31	28	26	38	37	36	41	41	41
low	high	med	2	2	2	32	29	27	39	38	37	42	42	42
low	high	high	3	3	3	33	30	28	40	39	38	43	43	43
med	low	low	2	2	2	31	28	26	42	41	40	45	45	45
med	low	med	5	5	5	32	29	27	42	41	40	45	45	45
med	low	high	9	9	9	33	30	28	42	41	40	45	45	45
med	med	low	2	2	2	32	29	27	40	39	38	43	43	43
med	med	med	3	3	3	32.5	29.5	27.5	41	40	39	44	44	44
med	med	high	5	5	5	33	30	28	42	41	40	45	45	45
med	high	low	1	1	1	32	29	27	39	38	37	42	42	42
med	high	med	2	2	2	33	30	28	40	39	38	43	43	43
med	high	high	3	3	3	34	31	29	41	40	39	44	44	44
high	low	low	2	2	2	32	29	27	43	42	41	46	46	46
high	low	med	5	5	5	33	30	28	43	42	41	46	46	46

high	low	high	9	9	9	34	31	29	43	42	41	46	46	46
high	med	low	2	2	2	33	30	28	41	40	39	44	44	44
high	med	med	3	3	3	33.5	30.5	28.5	42	41	40	45	45	45
high	med	high	5	5	5	34	31	29	43	42	41	46	46	46
high	high	low	1	1	1	33	30	28	40	39	38	43	43	43
high	high	med	2	2	2	34	31	29	41	40	39	44	44	44
high	high	high	3	3	3	35	32	30	42	41	40	45	45	45

C(TD, TI, N_t, F_t)

F_t

TD	TI	N_t	low	med	high
low	low	low	-3	-6	-21
low	low	med	-2	-5	-20
low	low	high	-1	-4	-19
low	med	low	-8	-11	-26
low	med	med	-7	-10	-25
low	med	high	-6	-9	-24
low	high	low	-18	-21	-36
low	high	med	-17	-20	-35
low	high	high	-16	-19	-34
med	low	low	-7	-10	-25
med	low	med	-6	-9	-24
med	low	high	-5	-8	-23
med	med	low	-12	-15	-30
med	med	med	-11	-14	-29
med	med	high	-10	-13	-28
med	high	low	-22	-25	-40
med	high	med	-21	-24	-39
med	high	high	-20	-23	-38
high	low	low	-13	-16	-31
high	low	med	-12	-15	-30
high	low	high	-11	-14	-29
high	med	low	-18	-21	-36
high	med	med	-17	-20	-35
high	med	high	-16	-19	-34
high	high	low	-28	-31	-46
high	high	med	-27	-30	-45
high	high	high	-26	-29	-44

B.2 Chapter 5

The simulation experiments presented in Chapter 5 makes use of a DBN in the user characteristics component. The parameters used in the DBN are defined below. The one CPT that we do not enumerate here is $Pr(MS_t|G, MS_{t-1}, EV_{t-1})$, representing the transition function of the goal model. This function is define according to the DFA presented in Section 5.2. We first show the CPTs that are independent of the machine state.

Pr(TN)

no	yes
.4	.6

Pr(NS_t|TN,DY_t)

TN	DY _t	low	med	high
no	easy	.7	.2	.1
yes	easy	.4	.3	.3
no	med	.4	.5	.1
yes	med	.1	.5	.4
no	hard	.3	.3	.4
yes	hard	.1	.2	.7

Pr(NS_t|TN,DY_t,NS_{t-1})

N _{t-1}	TN	DY _t	low	med	high
low	no	easy	.7	.2	.1
med	no	easy	.5	.3	.2
high	no	easy	.2	.2	.6
low	yes	easy	.4	.3	.3
med	yes	easy	.2	.3	.5
high	yes	easy	.1	.2	.7
low	no	med	.4	.5	.1
med	no	med	.3	.3	.4
high	no	med	.1	.2	.7
low	yes	med	.1	.5	.4
med	yes	med	.1	.2	.7
high	yes	med	.05	.15	.8
low	no	hard	.3	.3	.4
med	no	hard	.1	.3	.6
high	no	hard	.1	.1	.8

low	yes	hard	.1	.2	.7
med	yes	hard	.1	.1	.8
high	yes	hard	.05	.05	.9

Next, we define the parameters in the user characteristics component that are dependent on the machine state. Note that since the number of machine states increase as the number of observed goals are added to the library, these CPTs are dynamically generated according to the goals that are added.

The prior distribution for machine states, $Pr(MS_t|G)$, is used at the start of each episode. The probabilities are defined to be equally distributed across the start states of goal machines in the library. All machine states that are not the start state of the goal have zero probability.

$Pr(QUAL_t|MS_t, IS_t)$ is defined as $Pr(QUAL_t = N/A|MS_t, IS_t = off) = 1.0$ for all values of MS_t and $Pr(QUAL_t|MS_t, IS_t) = qualdistmat(MS_t, IS_t)$ for other values of IS_t . At a high level, depending on the distance between the current machine state and the help value offered, quality is defined accordingly. First, if MS_t is either at a special state (specifically, s0, the goal state, the accept state, and the reject state), then $Pr(QUAL_t|MS_t, IS_t)$ is defined as follows:

		Pr(QUAL_t MS_t, IS_t)			
MS_t	IS_t	N/A	low	med	high
s0	*	1.0	0	0	0
goal state	low	0	.8	.1	.1
goal state	med	0	.7	.2	.1
goal state	high	0	.6	.3	.1
accept	low	0	.8	.1	.1
accept	med	0	.7	.2	.1
accept	high	0	.6	.3	.1
reject	low	0	.8	.1	.1
reject	med	0	.7	.2	.1
reject	high	0	.6	.3	.1

Next, a simple look-up is done to determine the number of states between the machine state to the goal state of the corresponding machine. We refer to this value as d_t . Then, we define the remaining parameters of $Pr(QUAL_t|MS_t, IS_t)$ using the following:

		Pr(QUAL_t d_t, IS_t)			
d_t	IS_t	N/A	low	med	high
1-2	low	0	.4	.4	.2

1-2	med	0	.2	.6	.2
1-2	high	0	.2	.4	.4
3+	low	0	.1	.3	.6
3+	med	0	.1	.2	.7
3+	high	0	.1	.1	.8

The CPT for $Pr(DY_t|MS_t)$ is defined according to three cases based on MS_t . First, when MS_t is in the s0 state, we have:

Pr(DY_t d_t)				
MS_t	easy	med	hard	
s0	.8	.1	.1	

Otherwise, we define an intermediary variable d_t based on the number of states between MS_t and the goal state of the corresponding machine. In the second case, when MS_t is a non-extraneous state, we have:

Pr(DY_t d_t)				
d_t	easy	med	hard	
0-1	.6	.3	.1	
2	.2	.5	.3	
3,4	.1	.3	.6	

Lastly, assuming extra cognitive effort is required for users to recognize extraneous attributes, we define the following when MS_t is an extraneous state:

Pr(DY_t d_t)				
d_t	easy	med	hard	
1	.2	.5	.3	
2+	.1	.3	.6	

$Pr(EV_t|MS_t, NS_t, QUAL_t)$ is defined with an emphasis on the behavioural observation model of the user, so that MS_t and $QUAL_t$ play a minimal role in the definition. In particular, when MS_t is not at s0, entry events have a probability of zero. Also, when $QUAL_t = N/A$, all the response events have a probability of zero. The remaining event probabilities are defined in the following tables. Note that each event value is listed, starting with **sel** and ending with **misc**.

Pr(EV_t MS_t, QUAL_t=N/A, NS_t)										
MS_t	NS_t	sel	sel1	sel12	sel123	f1	f2	f3	f4	
s0	low	0.1082	0.1082	0.1082	0.1082	0.0022	0.0022	0.0022	0.0022	
s0	med	0.1174	0.1174	0.1174	0.1174	0.0023	0.0023	0.0023	0.0023	
s0	high	0.1029	0.1029	0.1029	0.1029	0.0021	0.0021	0.0021	0.0021	
other	low	0	0	0	0	0.0230	0.0230	0.0230	0.0230	

other	med	0	0	0	0	0.0234	0.0234	0.0234	0.0234
other	high	0	0	0	0	0.0169	0.0169	0.0169	0.0169
Pr(EV _t MS _t ,QUAL _t =N/A,NS _t)									
MS _t	NS _t	f5	f6	f7	f8	undo-1	undo-2	undo-3	undo-4
s0	low	0.0022	0.0022	0.0022	0.0022	0.0022	0.0022	0.0022	0.0022
s0	med	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023
s0	high	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021
other	low	0.0230	0.0230	0.0230	0.0230	0.0115	0.0115	0.0115	0.0115
other	med	0.0234	0.0234	0.0234	0.0234	0.0313	0.0313	0.0313	0.0313
other	high	0.0169	0.0169	0.0169	0.0169	0.0508	0.0508	0.0508	0.0508
Pr(EV _t MS _t ,QUAL _t =N/A,NS _t)									
MS _t	NS _t	undo-5	undo-6	undo-7	undo-8	pause	browse	waver	cons
s0	low	0.0022	0.0022	0.0022	0.0022	0.0043	0.0043	0.0043	0
s0	med	0.0023	0.0023	0.0023	0.0023	0.0704	0.0704	0.0704	0
s0	high	0.0021	0.0021	0.0021	0.0021	0.1646	0.1646	0.1646	0
other	low	0.0115	0.0115	0.0115	0.0115	0.0115	0.0115	0.0115	0
other	med	0.0313	0.0313	0.0313	0.0313	0.0313	0.0313	0.0313	0
other	high	0.0508	0.0508	0.0508	0.0508	0.1356	0.1356	0.1356	0
Pr(EV _t MS _t ,QUAL _t =N/A,NS _t)									
MS _t	NS _t	a1	a12	a123	a1234	a1a	a1ab	a1abc	a12a
s0	low	0	0	0	0	0	0	0	0
s0	med	0	0	0	0	0	0	0	0
s0	high	0	0	0	0	0	0	0	0
other	low	0	0	0	0	0	0	0	0
other	med	0	0	0	0	0	0	0	0
other	high	0	0	0	0	0	0	0	0
Pr(EV _t MS _t ,QUAL _t =N/A,NS _t)									
MS _t	NS _t	a12ab	a123a	click	type	misc			
s0	low	0	0	0.1732	0.1732	0.1732			
s0	med	0	0	0.0939	0.0939	0.0939			
s0	high	0	0	0.0206	0.0206	0.0206			
other	low	0	0	0.2299	0.2299	0.2299			
other	med	0	0	0.1563	0.1563	0.1563			
other	high	0	0	0.0169	0.0169	0.0169			
Pr(EV _t MS _t ,QUAL _t =low,NS _t)									
MS _t	NS _t	sel	sel1	sel12	sel123	f1	f2	f3	f4
s0	low	0.1057	0.1057	0.1057	0.1057	0.0021	0.0021	0.0021	0.0021
s0	med	0.1144	0.1144	0.1144	0.1144	0.0023	0.0023	0.0023	0.0023
s0	high	0.1006	0.1006	0.1006	0.1006	0.0020	0.0020	0.0020	0.0020
other	low	0	0	0	0	0.0280	0.0280	0.0280	0.0280
other	med	0	0	0	0	0.0215	0.0215	0.0215	0.0215

other	high	0	0	0	0	0.0084	0.0084	0.0084	0.0084
Pr(EV _t MS _t ,QUAL _t =low,NS _t)									
MS _t	NS _t	f5	f6	f7	f8	undo-1	undo-2	undo-3	undo-4
s0	low	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021
s0	med	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023
s0	high	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020
other	low	0.0280	0.0280	0.0280	0.0280	0.0093	0.0093	0.0093	0.0093
other	med	0.0215	0.0215	0.0215	0.0215	0.0215	0.0215	0.0215	0.0215
other	high	0.0084	0.0084	0.0084	0.0084	0.0252	0.0252	0.0252	0.0252
Pr(EV _t MS _t ,QUAL _t =low,NS _t)									
MS _t	NS _t	undo-5	undo-6	undo-7	undo-8	pause	browse	waver	cons
s0	low	0.0021	0.0021	0.0021	0.0021	0.0042	0.0042	0.0042	0.0021
s0	med	0.0023	0.0023	0.0023	0.0023	0.0686	0.0686	0.0686	0.0023
s0	high	0.0020	0.0020	0.0020	0.0020	0.1610	0.1610	0.1610	0.0020
other	low	0.0093	0.0093	0.0093	0.0093	0.0093	0.0093	0.0093	0.0187
other	med	0.0215	0.0215	0.0215	0.0215	0.0215	0.0215	0.0215	0.0538
other	high	0.0252	0.0252	0.0252	0.0252	0.0672	0.0672	0.0672	0.0840
Pr(EV _t MS _t ,QUAL _t =low,NS _t)									
MS _t	NS _t	a1	a12	a123	a1234	a1a	a1ab	a1abc	a12a
s0	low	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021
s0	med	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023
s0	high	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020
other	low	0.0093	0.0093	0.0093	0.0093	0.0093	0.0093	0.0093	0.0093
other	med	0.0215	0.0215	0.0215	0.0215	0.0215	0.0215	0.0215	0.0215
other	high	0.0420	0.0420	0.0420	0.0420	0.0420	0.0420	0.0420	0.0420
Pr(EV _t MS _t ,QUAL _t =low,NS _t)									
MS _t	NS _t	a12ab	a123a	click	type	misc			
s0	low	0.0021	0.0021	0.1691	0.1691	0.1691			
s0	med	0.0023	0.0023	0.0915	0.0915	0.0915			
s0	high	0.0020	0.0020	0.0201	0.0201	0.0201			
other	low	0.0093	0.0093	0.1869	0.1869	0.1869			
other	med	0.0215	0.0215	0.1075	0.1075	0.1075			
other	high	0.0420	0.0420	0.0084	0.0084	0.0084			
Pr(EV _t MS _t ,QUAL _t =med,NS _t)									
MS _t	NS _t	sel	sel1	sel12	sel123	f1	f2	f3	f4
s0	low	0.1057	0.1057	0.1057	0.1057	0.0021	0.0021	0.0021	0.0021
s0	med	0.1144	0.1144	0.1144	0.1144	0.0023	0.0023	0.0023	0.0023
s0	high	0.1006	0.1006	0.1006	0.1006	0.0020	0.0020	0.0020	0.0020
other	low	0	0	0	0	0.0176	0.0176	0.0176	0.0176
other	med	0	0	0	0	0.0104	0.0104	0.0104	0.0104
other	high	0	0	0	0	0.0044	0.0044	0.0044	0.0044

		Pr(EV _t MS _t ,QUAL _t =med,NS _t)									
MS _t	NS _t	f5	f6	f7	f8	undo-1	undo-2	undo-3	undo-4		
s0	low	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021		
s0	med	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023		
s0	high	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020		
other	low	0.0176	0.0176	0.0176	0.0176	0.0059	0.0059	0.0059	0.0059		
other	med	0.0104	0.0104	0.0104	0.0104	0.0104	0.0104	0.0104	0.0104		
other	high	0.0044	0.0044	0.0044	0.0044	0.0131	0.0131	0.0131	0.0131		
		Pr(EV _t MS _t ,QUAL _t =med,NS _t)									
MS _t	NS _t	undo-5	undo-6	undo-7	undo-8	pause	browse	waver	cons		
s0	low	0.0021	0.0021	0.0021	0.0021	0.0042	0.0042	0.0042	0.0021		
s0	med	0.0023	0.0023	0.0023	0.0023	0.0686	0.0686	0.0686	0.0023		
s0	high	0.0020	0.0020	0.0020	0.0020	0.1610	0.1610	0.1610	0.0020		
other	low	0.0059	0.0059	0.0059	0.0059	0.0059	0.0059	0.0059	0.0294		
other	med	0.0104	0.0104	0.0104	0.0104	0.0104	0.0104	0.0104	0.0777		
other	high	0.0131	0.0131	0.0131	0.0131	0.0349	0.0349	0.0349	0.0873		
		Pr(EV _t MS _t ,QUAL _t =med,NS _t)									
MS _t	NS _t	a1	a12	a123	a1234	a1a	a1ab	a1abc	a12a		
s0	low	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021		
s0	med	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023		
s0	high	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020		
other	low	0.0059	0.0059	0.0059	0.0059	0.0059	0.0059	0.0059	0.0059		
other	med	0.0415	0.0415	0.0415	0.0415	0.0415	0.0415	0.0415	0.0415		
other	high	0.0655	0.0655	0.0655	0.0655	0.0655	0.0655	0.0655	0.0655		
		Pr(EV _t MS _t ,QUAL _t =med,NS _t)									
MS _t	NS _t	a12ab	a123a	click	type	misc					
s0	low	0.0021	0.0021	0.1691	0.1691	0.1691					
s0	med	0.0023	0.0023	0.0915	0.0915	0.0915					
s0	high	0.0020	0.0020	0.0201	0.0201	0.0201					
other	low	0.0059	0.0059	0.2353	0.2353	0.2353					
other	med	0.0415	0.0415	0.1036	0.1036	0.1036					
other	high	0.0655	0.0655	0.0044	0.0044	0.0044					
		Pr(EV _t MS _t ,QUAL _t =high,NS _t)									
MS _t	NS _t	sel	sel1	sel12	sel123	f1	f2	f3	f4		
s0	low	0.1057	0.1057	0.1057	0.1057	0.0021	0.0021	0.0021	0.0021		
s0	med	0.1144	0.1144	0.1144	0.1144	0.0023	0.0023	0.0023	0.0023		
s0	high	0.1006	0.1006	0.1006	0.1006	0.0020	0.0020	0.0020	0.0020		
other	low	0	0	0	0	0.0117	0.0117	0.0117	0.0117		
other	med	0	0	0	0	0.0057	0.0057	0.0057	0.0057		
other	high	0	0	0	0	0.0020	0.0020	0.0020	0.0020		
		Pr(EV _t MS _t ,QUAL _t =high,NS _t)									

MS_t	NS_t	f5	f6	f7	f8	undo-1	undo-2	undo-3	undo-4
s0	low	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021
s0	med	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023
s0	high	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020
other	low	0.0117	0.0117	0.0117	0.0117	0.0029	0.0029	0.0029	0.0029
other	med	0.0057	0.0057	0.0057	0.0057	0.0057	0.0057	0.0057	0.0057
other	high	0.0020	0.0020	0.0020	0.0020	0.0099	0.0099	0.0099	0.0099
Pr(EV_t MS_t,QUAL_t=high,NS_t)									
MS_t	NS_t	undo-5	undo-6	undo-7	undo-8	pause	browse	waver	cons
s0	low	0.0021	0.0021	0.0021	0.0021	0.0042	0.0042	0.0042	0.0021
s0	med	0.0023	0.0023	0.0023	0.0023	0.0686	0.0686	0.0686	0.0023
s0	high	0.0020	0.0020	0.0020	0.0020	0.1610	0.1610	0.1610	0.0020
other	low	0.0029	0.0029	0.0029	0.0029	0.0029	0.0029	0.0029	0.0292
other	med	0.0057	0.0057	0.0057	0.0057	0.0057	0.0057	0.0057	0.0575
other	high	0.0099	0.0099	0.0099	0.0099	0.0158	0.0158	0.0158	0.0594
Pr(EV_t MS_t,QUAL_t=high,NS_t)									
MS_t	NS_t	a1	a12	a123	a1234	a1a	a1ab	a1abc	a12a
s0	low	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021
s0	med	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023	0.0023
s0	high	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020
other	low	0.0146	0.0146	0.0146	0.0146	0.0146	0.0146	0.0146	0.0146
other	med	0.0575	0.0575	0.0575	0.0575	0.0575	0.0575	0.0575	0.0575
other	high	0.0792	0.0792	0.0792	0.0792	0.0792	0.0792	0.0792	0.0792
Pr(EV_t MS_t,QUAL_t=high,NS_t)									
MS_t	NS_t	a12ab	a123a	click	type	misc			
s0	low	0.0021	0.0021	0.1691	0.1691	0.1691			
s0	med	0.0023	0.0023	0.0915	0.0915	0.0915			
s0	high	0.0020	0.0020	0.0201	0.0201	0.0201			
other	low	0.0146	0.0146	0.2332	0.2332	0.2332			
other	med	0.0575	0.0575	0.0862	0.0862	0.0862			
other	high	0.0792	0.0792	0.0020	0.0020	0.0020			

While the event vocabulary includes entry events that involve extraneous attributes (i.e., `sel1a`, `sel1ab`, `sel1abc`, `sel12a`, `sel12ab`, `sel123a`), the actual implementation does not model these transitions. Instead, the event post-processing step that modifies the event history as described in Table 5.5 to Table 5.9 is used to handle these cases where the user changes previously completed goals.

Lastly, the action selection component in the simulation experiments make use of $R(JEU, NS)$ and $C(NS)$, defined as follows:

NS	R(JEU,NS)	C(NS)
low	0.5 * JEU	4.0
med	JEU	2.0
high	2.0 * JEU	1.0

B.3 Chapter 6 Case Study Two

The parameters used in the MDP in Section 6.6 are defined as follows. Specifically, the probability distribution of our approximate Savings calculation is defined in terms of the number of menu items shown ($\#Shn$):

Pr(Svgs #Shn)				
#Shn	low	med	high	
1	.125	.125	.750	
2	.250	.125	.625	
3	.375	.125	.500	
4	.500	.125	.375	
5	.625	.125	.250	
6	.750	.125	.125	

The transition function for tolerance, independence, frustration, and neediness are defined as an identity function. The transition function for distractibility is defined in terms of distractibility at the previous time step and the number of menu items shown ($\#Shn$).

Pr(D _t D _{t-1} ,#Shn)				
D _{t-1}	#Shn	low	med	high
low	1-2	.9	.09	.01
med	1-2	.8	.19	.01
high	1-2	.7	.29	.01
low	3-4	.7	.2	.1
med	3-4	.3	.4	.3
high	3-4	.1	.3	.7
low	5-6	.2	.7	.1
med	5-6	.1	.2	.7
high	5-6	.0	.1	.9

The parameters for the reward function is defined as $R = -Bloat + Svgs$. Specifically, we have $Bloat = 3.5 \times D \times \exp(XS/2)$ for feature-shy users, where D is distractibility

and XS is the number of menu items shown ($\#Shn$) minus the number of menu items used. For feature-keen users, we simply use $Bloat = D \times XS$. In the first case where the experiment was designed to assess bloat only, we define savings as $Svgs = 10 \times Qual$, where $Qual$ is the selection time savings by taking the system's adaptive suggestion. This variable is coarsely defined as no suggestion, low quality, medium quality, and high quality. In the second case where the experiment modeled the impact on savings according to different user states, we have $Svgs = 2.5 \times N \times Qual - F \times Qual + I \times Qual$, where $Qual$ is defined in the same way as before, N is the user's neediness level, F is frustration level, and I is the independence level. For D, F, N, I , low is indexed as 1, medium is 2, and high is 3. For Tol , feature-keen is indexed as 1, and feature-shy is 2.

Appendix C

Full Results

In this appendix, we document the results for all the user types in Chapter 4.

C.1 Chapter 4 Case Study One Results

A selected sample of average rewards was reported in Section 4.3. Here, we report the average rewards for all 36 user types.

C.2 Chapter 4 Case Study One Last M% Results

A selected sample of average rewards for the last 10% and last 2% of the data were reported in Section 4.3. Here, we report the average rewards for those portions of the data for all 36 user types.

Table C.1: Comparison of policies using average rewards by user profile $\{TD, TI, TN, TF\}$

User Type			ALWAYS	MEU	THRESH	NEVER
1	$\{1,1,1,1\}$	min, max	-19, 46	-19, 46	-19, 46	0, 0
		avg	-2.82	0.25	1.31	0
		stderr	0.4619	1.3031	1.0020	0
2	$\{1,1,1,2\}$	min, max	-19, 46	-19, 46	-19, 46	0, 0
		avg	-7.52	-4.66	-3.84	0
		stderr	0.3557	1.2498	1.0975	0
3	$\{1,1,2,1\}$	min, max	-19, 46	-19, 46	-19, 46	0, 0
		avg	-1.78	0.90	2.45	0
		stderr	0.6016	1.2492	1.2135	0
4	$\{1,1,2,2\}$	min, max	-19, 46	-19, 46	-19, 46	0, 0
		avg	-6.35	-3.28	-2.40	0
		stderr	0.3512	0.8724	1.1979	0
5	$\{2,1,1,1\}$	min, max	-22, 41	-22, 41	-22, 41	0, 0
		avg	-6.59	-4.20	-3.08	0
		stderr	0.4679	0.5396	1.0697	0
6	$\{2,1,1,2\}$	min, max	-22, 41	-22, 41	-22, 41	0, 0
		avg	-10.84	-8.89	-7.79	0
		stderr	0.3487	0.8573	0.6750	0
7	$\{2,1,2,1\}$	min, max	-22, 41	-22, 41	-22, 41	0, 0
		avg	-5.53	-3.04	-1.47	0
		stderr	0.5049	1.2870	0.8924	0
8	$\{2,1,2,2\}$	min, max	-22, 41	-22, 41	-22, 41	0, 0
		avg	-9.73	-7.39	-6.75	0
		stderr	0.4278	0.7861	0.6972	0

Table C.2: Comparison of policies using average rewards by user profile $\{TD, TI, TN, TF\}$

User Type			ALWAYS	MEU	THRESH	NEVER
9	$\{3,1,1,1\}$	min, max	-29, 39	-29, 39	-29, 39	0, 0
		avg	-12.99	-9.25	-9.25	0
		stderr	0.3335	2.9647	1.1559	0
10	$\{3,1,1,2\}$	min, max	-29, 39	-29, 39	-29, 39	0, 0
		avg	-17.63	-13.37	-14.73	0
		stderr	0.2108	2.0658	0.7139	0
11	$\{3,1,2,1\}$	min, max	-29, 39	-29, 39	-29, 39	0, 0
		avg	-11.47	-7.73	-7.68	0
		stderr	0.4799	2.1944	1.2173	0
12	$\{3,1,2,2\}$	min, max	-29, 39	-29, 39	-29, 39	0, 0
		avg	-16.23	-10.96	-12.85	0
		stderr	0.3097	1.9442	0.6908	0
13	$\{1,2,1,1\}$	min, max	-22, 45	-22, 45	-22, 45	0, 0
		avg	-4.71	-1.05	-0.72	0
		stderr	0.5518	1.8969	1.3593	0
14	$\{1,2,1,2\}$	min, max	-22, 45	-22, 45	-22, 45	0, 0
		avg	-9.67	-6.57	-6.07	0
		stderr	0.1799	0.9516	1.2208	0
15	$\{1,2,2,1\}$	min, max	-22, 45	-22, 45	-22, 45	0, 0
		avg	-3.93	-0.42	0.75	0
		stderr	0.4304	1.3379	1.4287	0
16	$\{1,2,2,2\}$	min, max	-22, 45	-22, 45	-22, 45	0, 0
		avg	-8.52	-5.48	-4.39	0
		stderr	0.3935	0.9346	0.6547	0

Table C.3: Comparison of policies using average rewards by user profile {TD, TI, TN, TF}

User Type			ALWAYS	MEU	THRESH	NEVER
17	{2,2,1,1}	min, max	-25, 40	-25, 40	-25, 40	0, 0
		avg	-8.92	-5.68	-5.07	0
		stderr	0.3337	0.8886	0.9980	0
18	{2,2,1,2}	min, max	-25, 40	-25, 40	-25, 40	0, 0
		avg	-13.26	-9.90	-10.10	0
		stderr	0.2940	1.1613	0.9138	0
19	{2,2,2,1}	min, max	-25, 40	-25, 40	-25, 40	0, 0
		avg	-7.75	-4.49	-4.18	0
		stderr	0.5381	1.1799	1.4625	0
20	{2,2,2,2}	min, max	-25, 40	-25, 40	-25, 40	0, 0
		avg	-12.05	-8.62	-8.37	0
		stderr	0.3041	1.0935	0.7984	0
21	{3,2,1,1}	min, max	-32, 38	-32, 38	-32, 38	0, 0
		avg	-15.30	-7.48	-11.86	0
		stderr	0.4425	4.7051	1.0875	0
22	{3,2,1,2}	min, max	-32, 38	-32, 33	-32, 38	0, 0
		avg	-20.05	-1.03	-16.56	0
		stderr	0.3279	14.1416	0.8814	0
23	{3,2,2,1}	min, max	-32, 38	-32, 38	-32, 38	0, 0
		avg	-14.11	-9.25	-9.84	0
		stderr	0.5043	2.0599	1.1017	0
24	{3,2,2,2}	min, max	-32, 38	-32, 38	-32, 38	0, 0
		avg	-18.66	2.04	-15.32	0
		stderr	0.2724	12.1804	1.1808	0

Table C.4: Comparison of policies using average rewards by user profile {TD, TI, TN, TF}

User Type			ALWAYS	MEU	THRESH	NEVER
25	{1,3,1,1}	min, max	-33, 44	-33, 44	-33, 44	0, 0
		avg	-11.03	-5.10	-4.50	0
		stderr	0.8988	1.2277	1.6504	0
26	{1,3,1,2}	min, max	-33, 44	-33, 44	-33, 44	0, 0
		avg	-16.06	-8.30	-10.84	0
		stderr	0.7687	2.2112	1.0073	0
27	{1,3,2,1}	min, max	-33, 44	-33, 44	-33, 44	0, 0
		avg	-10.23	-4.68	-4.49	0
		stderr	0.5279	0.9566	1.5662	0
28	{1,3,2,2}	min, max	-33, 44	-33, 44	-33, 44	0, 0
		avg	-15.52	-8.91	-10.45	0
		stderr	0.6357	1.6503	1.3182	0
29	{2,3,1,1}	min, max	-36, 39	-36, 39	-36, 39	0, 0
		avg	-15.35	-12.85	-10.45	0
		stderr	0.4831	10.9887	1.3897	0
30	{2,3,1,2}	min, max	-36, 39	-36, 36	-36, 39	0, 0
		avg	-20.33	-2.30	-15.32	0
		stderr	0.5081	19.2724	1.9738	0
31	{2,3,2,1}	min, max	-36, 39	-36, 39	-36, 39	0, 0
		avg	-14.87	-7.01	-9.61	0
		stderr	0.5552	9.4216	0.9182	0
32	{2,3,2,2}	min, max	-36, 39	-36, 37	-36, 39	0, 0
		avg	-19.53	-7.47	-14.38	0
		stderr	0.6669	17.9949	0.7990	0

Table C.5: Comparison of policies using average rewards by user profile {TD, TI, TN, TF}

User Type			ALWAYS	MEU	THRESH	NEVER
33	{3,3,1,1}	min, max	-43, 37	-43, 35	-43, 37	0, 0
		avg	-22.03	-7.24	-17.63	0
		stderr	0.7254	19.4154	1.0563	0
34	{3,3,1,2}	min, max	-43, 37	-43, 35	-43, 37	0, 0
		avg	-26.66	-0.41	-21.69	0
		stderr	0.6841	23.9873	1.8065	0
35	{3,3,2,1}	min, max	-43, 37	-41, 37	-43, 37	0, 0
		avg	-21.41	2.91	-15.86	0
		stderr	0.5892	16.6874	1.4483	0
36	{3,3,2,2}	min, max	-43, 37	-43, 35	-43, 37	0, 0
		avg	-26.16	-1.63	-21.27	0
		stderr	0.6207	21.4522	1.0899	0
Average			-12.93	-5.35	-8.73	0

Table C.6: Comparison of policies using the last 10% of data for average rewards by user profile $\{TD, TI, TN, TF\}$

User Type	ALWAYS	MEU	THRESH	NEVER
1 {1,1,1,1}	-2.9141	0.7652	2.1966	0
2 {1,1,1,2}	-7.7373	-4.2218	-3.2680	0
3 {1,1,2,1}	-1.6432	0.5362	2.4397	0
4 {1,1,2,2}	-6.3825	-3.3902	-2.1466	0
5 {2,1,1,1}	-6.1708	-2.0422	-3.1841	0
6 {2,1,1,2}	-11.1220	-8.4090	-8.0539	0
7 {2,1,2,1}	-5.3690	-2.8429	-2.0344	0
8 {2,1,2,2}	-9.3995	-5.8470	-6.0682	0
9 {3,1,1,1}	-12.6404	-5.7767	-10.0337	0
10 {3,1,1,2}	-17.8410	7.8760	-14.2073	0
11 {3,1,2,1}	-11.2644	-3.8460	-7.8415	0
12 {3,1,2,2}	-15.9388	-2.7027	-13.2700	0
13 {1,2,1,1}	-4.5190	-0.5221	-0.6942	0
14 {1,2,1,2}	-9.2880	-5.2276	-5.0077	0
15 {1,2,2,1}	-3.9625	-1.1946	1.5964	0
16 {1,2,2,2}	-8.1086	-5.7097	-3.3728	0
17 {2,2,1,1}	-9.1030	-3.7393	-5.2369	0
18 {2,2,1,2}	-13.2699	-9.0909	-10.0918	0

Table C.7: Comparison of policies using the last 10% of data for average rewards by user profile $\{TD, TI, TN, TF\}$

	User Type	ALWAYS	MEU	THRESH	NEVER
19	$\{2,2,2,1\}$	-7.3478	-2.6078	-5.2405	0
20	$\{2,2,2,2\}$	-12.1273	-7.7074	-7.8564	0
21	$\{3,2,1,1\}$	-15.6988	4.2318	-12.3889	0
22	$\{3,2,1,2\}$	-19.9589	0.0000	-16.5983	0
23	$\{3,2,2,1\}$	-13.8764	2.2515	-9.9175	0
24	$\{3,2,2,2\}$	-18.2497	0.0000	-14.6149	0
25	$\{1,3,1,1\}$	-10.8117	-3.4823	-2.6227	0
26	$\{1,3,1,2\}$	-15.9728	-7.3480	-10.5416	0
27	$\{1,3,2,1\}$	-10.4011	-2.7771	-4.1220	0
28	$\{1,3,2,2\}$	-15.4257	-7.5042	-10.9083	0
29	$\{2,3,1,1\}$	-14.9679	0.0000	-11.8224	0
30	$\{2,3,1,2\}$	-20.0897	0.0000	-15.6890	0
31	$\{2,3,2,1\}$	-14.7549	0.0000	-10.4257	0
32	$\{2,3,2,2\}$	-19.5911	0.0000	-15.1013	0
33	$\{3,3,1,1\}$	-21.1165	0.0000	-18.2323	0
34	$\{3,3,1,2\}$	-26.5237	0.0000	-22.6452	0
35	$\{3,3,2,1\}$	-21.1625	0.0000	-16.8875	0
36	$\{3,3,2,2\}$	-25.7536	0.0000	-22.6358	0

Table C.8: Comparison of policies using the last 2% of data for average rewards by user profile $\{TD, TI, TN, TF\}$

	User Type	ALWAYS	MEU	THRESH	NEVER
1	$\{1,1,1,1\}$	-3.8911	3.5355	6.3547	0
2	$\{1,1,1,2\}$	-8.1778	-0.1955	-1.6511	0
3	$\{1,1,2,1\}$	-1.9776	0.8721	3.4679	0
4	$\{1,1,2,2\}$	-7.2468	-0.3346	1.5320	0
5	$\{2,1,1,1\}$	-6.9467	4.4297	-3.8022	0
6	$\{2,1,1,2\}$	-11.3413	-4.9099	-2.0727	0
7	$\{2,1,2,1\}$	-6.5703	2.1446	3.0228	0
8	$\{2,1,2,2\}$	-9.0370	2.3187	-4.8436	0
9	$\{3,1,1,1\}$	-12.6540	9.4056	-4.9120	0
10	$\{3,1,1,2\}$	-17.5212	1.2593	-7.3382	0
11	$\{3,1,2,1\}$	-12.9457	2.8563	-5.5261	0
12	$\{3,1,2,2\}$	-16.2653	1.7278	-11.4265	0
13	$\{1,2,1,1\}$	-5.0400	2.1391	-1.1569	0
14	$\{1,2,1,2\}$	-8.6513	-4.4883	-4.1381	0
15	$\{1,2,2,1\}$	-4.8716	2.9084	4.2107	0
16	$\{1,2,2,2\}$	-9.1713	-4.8738	1.9499	0
17	$\{2,2,1,1\}$	-8.6034	0.9451	-4.6811	0
18	$\{2,2,1,2\}$	-13.7882	-7.8220	-9.2747	0

Table C.9: Comparison of policies using the last 2% of data for average rewards by user profile $\{TD, TI, TN, TF\}$

	User Type	ALWAYS	MEU	THRESH	NEVER
19	$\{2,2,2,1\}$	-7.3502	3.4158	-3.7034	0
20	$\{2,2,2,2\}$	-12.4338	-5.8740	-4.2261	0
21	$\{3,2,1,1\}$	-15.6034	-1.0972	-12.6617	0
22	$\{3,2,1,2\}$	-20.6203	0.0000	-12.8771	0
23	$\{3,2,2,1\}$	-13.6310	3.7412	-9.5284	0
24	$\{3,2,2,2\}$	-19.2044	0.0000	-11.4283	0
25	$\{1,3,1,1\}$	-12.1692	0.7639	-1.7395	0
26	$\{1,3,1,2\}$	-17.3464	-0.1498	-6.4788	0
27	$\{1,3,2,1\}$	-10.9105	-5.0104	-2.4562	0
28	$\{1,3,2,2\}$	-16.3158	-4.4624	-7.4611	0
29	$\{2,3,1,1\}$	-15.3193	0.0000	-9.0376	0
30	$\{2,3,1,2\}$	-21.5043	0.0000	-14.9816	0
31	$\{2,3,2,1\}$	-14.7095	0.0000	-10.3441	0
32	$\{2,3,2,2\}$	-20.6389	0.0000	-12.2483	0
33	$\{3,3,1,1\}$	-20.4937	0.0000	-19.6408	0
34	$\{3,3,1,2\}$	-27.1441	0.0000	-15.3716	0
35	$\{3,3,2,1\}$	-21.9023	0.0000	-16.6290	0
36	$\{3,3,2,2\}$	-25.0868	0.0000	-20.3291	0