Intelligent User Interfaces: Modelling the User

Marcus Rohrbach and **Uwe Schmidt**
rohrbach@cs.ubc.ca and uschmidt@cs.ubc.ca
Artificial Intelligence 1: CPSC 502
Computer Science
University of British Columbia

Abstract

Current user interfaces are very complex with a large number of functions. *Intelligent* user interfaces try to help the users performing their tasks. We describe how (Horvitz *et al.* 1998) construct and created a user model to meet the different challenges which exist. Building on this, we first look how (Hui & Boutilier 2006) model the user's features explicitly to suggest help based on the user's state. Secondly, we explore how (Shen *et al.* 2006) predict the user's current task.

Introduction

Current user interfaces (UIs) are often "bloated" (McGrenere, Baecker, & Booth 2002) and user tasks are getting more complex and more complicated. In these cases artificial intelligence can help to reduce the "bloat" in user interfaces and assist the user.

However, several challenges arise when adapting the user interface or providing help: Interpreting user's activities and predicting goals correctly is very difficult because it is many faceted and the system has only a limited amount of observations. These consist normally of the user's interaction with the interface. Additionally the decisions have to be made online in a timely manner. For these predictions a model has to be created in most cases. Here it is difficult to determine which variables depend on others and in which way they influence each other. Once a model is created, the model's parameters have to be learned and in some cases missing data to complete the model is a further difficulty.

Structure of Intelligent UIs

In general intelligent user interfaces are structured as shown in figure 1. We have a model of arbitrary complexity which observers the user interfaces, i.e. the user interaction with the computer. Based on prior knowledge (included in the model) and previous and current observations it decides how to adapt or affect the user interface for the user. A user model can consist of a large number of facets. We will discuss in this paper two of the most import ones in detail. On the one hand there are the users features and needs and on the other hand the user's goals and tasks, i.e. what the user is doing and wants to achieve. First we will look on a more general approach on how to build user models. Then we will focus first on *features and needs* and then on *goals and tasks*.

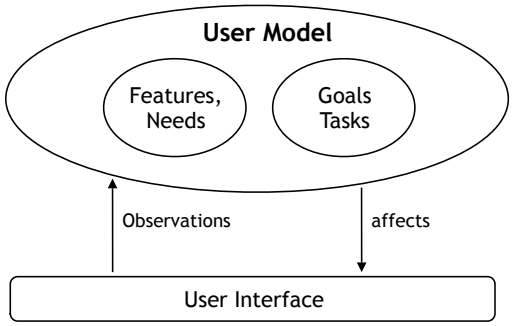

Figure 1: Relation of user model and user interface

User Modelling and Deducing Goals

In this section we describe an approach how to model the user as outlined by Microsoft Research in the Lumière Project (Horvitz *et al.* 1998).

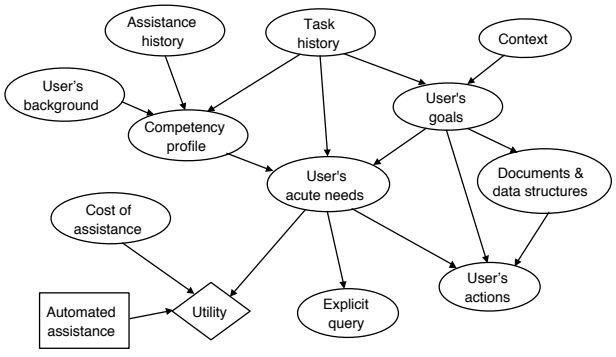

Figure 2: Decision network to deduce whether help should be provided (Horvitz *et al.* 1998)

Decision Network for Assistance. Horvitz et al. experimented with automated assistance, i.e. which information to present to the user, how and when to help the user reaching their goal. They built a decision network for deciding if the user should receive assistance. As shown in figure 2

the utility depends on the value of the *automated assistance*, the *cost of the assistance* due to interrupting the user, and the *user's acute needs*. The latter one depends on variables, such as the *user's goals* and the *task history*, and influences for example the *user's actions*. A threshold defines if the utility is high enough to assist the user.

Creating a Model. They set up a user study with a Wizard of Oz[1] design to find out what is important when creating such a model. Subjects were asked to perform a certain task on a desktop computer and were told that they would receive help on a separate screen. Experts were placed in another room seeing only the subjects interface and the interaction of the subject with it. With these information they were asked to give help to the subjects by typing recommendations which were shown on the subjects second screen.

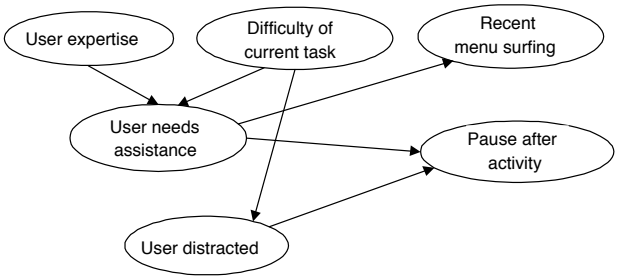

Figure 3: Bayesian Network to model when assistance is needed (Horvitz *et al.* 1998)

The experiment showed that it is possible to identify user goals and needs, but it is difficult. The experts did need some time of observation until they could predict more precisely what the user wants to do. Another major result was, that if incorrect or poor advice is given, a high cost is associated with that because users tried to follow that advice which in turn gave the expert a wrong confirmation.

Additionally they were able to deduce a model when assistance is needed. Figure 3 displays a part of the relationships. On the one hand, whether the *user needs assistance* depends on features like the *users expertise* and the *difficulty of the task*. On the other hand the *user's needs* influence the behaviour of the user, e.g. *recent menu surfing* or *pausing after an activity* show that they are not sure what to do. This can be observed and a system can infer if the user needs assistance.

Time Dependency. The models described so far did not take into account that variables change over time and that they depend on previous states. For example the users' goals change, because users change their tasks. Such behaviour can be modelled by a dynamic Bayesian network. It is able to model variables which change over time and depend on previous states, such as the goal in our example. As can be seen in figure 4 the goal depends on the previous goal, which again depends on previous instances of the variable goal. However, there might also be variables which remain constant over time, such as the user profile, or variables which change over time but do not depend on their previous state, such as observation, shown as *E* in the figure.

[1] *Wizard of Oz* design means that a system is emulated by a human

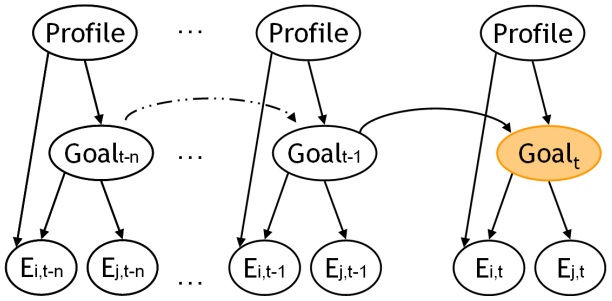

Figure 4: Dynamic Bayesian Network to model goals depending on previous states; based on (Horvitz *et al.* 1998)

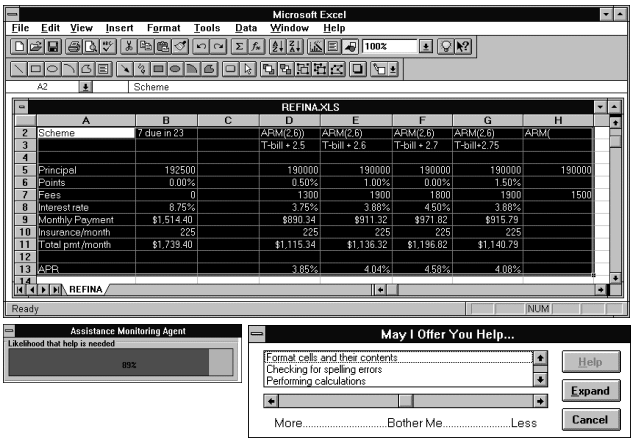

Figure 5: Lumière Project: Excel Prototype (Horvitz *et al.* 1998)

Lumière Prototype vs. MS Office '97 Assistant. A prototype for Microsoft Excel was created (figure 5) in the Lumière project. In the prototype the previously discussed ideas were included. (Horvitz *et al.* 1998) did not state if and how the prototype was evaluated, but they discuss how Microsoft transferred some of these ideas to the Office '97 Assistant. Similar to the prototype a large range of observations of the current view and document were used to evaluate the user's goals and needs. However, the assistance is not based on a model which determines if the user needs assistance. Neither the user expertise is considered nor is a persistent user profile established. In the Office '97 Assistant thousands of goals are considered, but only with a shallow model. All these reductions led to a assistant which is not very sophisticated anymore.

Inferring User Features

The Lumière project, discussed in the previous section, tried to infer the user's neediness. In addition to that, (Hui & Boutilier 2006) are aiming to infer the user's state in terms of user features, which includes emotional state and personality, to adapt the user interface to them. They are modelling the user's features explicitly and then try to infer these features through observations of user behaviour.

User Model. Users are modelled in terms of their attitudes, which are transient over time, and their personality traits, which are assumed to be static over time. All these user parameters are random variables, because of the uncertainty of the user's needs and preferences. The user state and type are composed of the user's attitudes and personality traits:

User state:
- Attitudes
  - Frustration (F)
  - Neediness (N)
- Personality traits
  - Tendency to get distracted (TD)
  - Tendency to work independently (TI)
  - Tendency towards Frustration (TF)
  - Tendency towards Neediness (TN)

User type

All variables of the user's state have a discrete domain of $\{1, 2, 3\}$, where 1 = *not*, 2 = *somewhat* and 3 = *very* (e.g. $N = 2$ means that the user is somewhat in need of help at the moment). TF and TN can only take boolean values, e.g. the user either has a tendency towards frustration or not. After all, there are 36 user types and 81 user states.

They model the user's static features with an eye towards a persistent user profile which may be transferred between different applications.

The system maintains a probability distribution over all user states, which is updated every time a new observation about the user's behaviour enters the system. The system's actions are based on the belief that a user is in a particular state (i.e. has certain features at the moment).

Causality & Inference. The users' acceptance of help depends on the quality of the available help, the users' tendency to work independently and whether they consider help. The users' consideration of help is dependent on the users' state $\{F, N, TD, TI\}$. These influential factors can be seen in figure 6.

Keeping these influential factors in mind, they model their system with a Dynamic Bayesian Network (DBN) in causal order, which can be seen in figure 7. The user's frustration, neediness and consideration of help at a certain point in time is dependent on its value at the last time step (coloured orange). The user's type $\{TF, TN, TD, TI\}$ does not change over time.

The observations are drawn with double lines. The DBN observes the users' behaviour (OBS), which is causally dependent on their current state; the available help and its quality as well as the direct request of the user for more or fewer help (SYS).

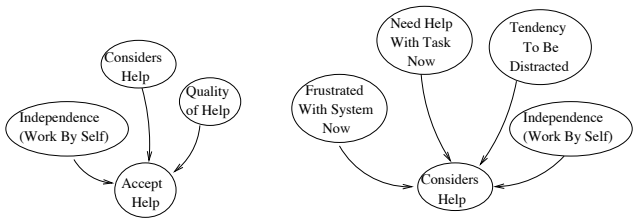

Figure 6: Influential factors (Hui & Boutilier 2006)

Exact inference in the DBN is done with a *clique tree* algorithm. This family of algorithms "is particularly suited to the case where observations arrive incrementally, where we want the posterior probability of each node, and where the cost of building the clique tree can be amortized over many cases." (Zhang & Poole 1996).

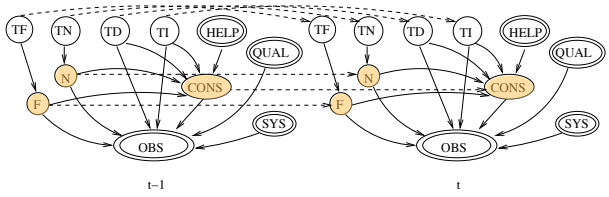

Figure 7: Dynamic Bayesian Network (Hui & Boutilier 2006)

Observations. The observation model is domain specific, but fully observable variables for F, N, TD, TI have to be derived from the user's observations. In their paper they use the domain of text-editing where word prediction is considered as automated help (i.e. the user is typing text and the system tries to suggest the most probable word the user wants to type). In this domain, the observation function for the user's frustration can for instance be modelled as observing fast mouse movements and a rapid sequence of clicks as well as the user jamming into the keyboard.

Decision Policy. Based on all observations that go into the DBN, a decision has to be made at each time whether to suggest help to the user ($sHelp$) or not. The system reasons about the rewards and costs of its actions, based on the current belief in the user's state. The user can either accept (acc) or reject ($\neg acc$) the suggested help. The expected utility (EU) for suggesting help:

$$\begin{aligned} EU(sHelp) &= EU(sHelp|acc)P(acc) \\ &+ EU(sHelp|\neg acc)P(\neg acc) \end{aligned}$$

The expected utility for not suggesting help is calculated analogously. The system then carries out the action with the maximum expected utility, i.e. suggest help if $EU(sHelp) > EU(\neg sHelp)$.

Simulation. They evaluated their user model by running simulations with their text-editing domain. The user's behaviour was simulated by sampling from the DBN. On average, their system was able to determine the correct user type based on the simulated observations for all possible user types.

Learning Model Parameters. After evaluating their user model, they learned the parameters for the DBN (prior distributions, transition functions and observation functions) by conducting experiments with 45 users. They recorded the user's behaviour and acquired the user's features through questionnaires. After they collected all the data, they used the *Expectation Maximation* algorithm to learn the parameters for the DBN. The reason behind this parameter learning was to replace their initially handcrafted parameters by parameters that have been derived from real user behaviour.

User Study and Results. To validate their learned model, they made a usability experiment with 4 participants by comparing their adaptive policy MEU[2] against the static policies ALWAYS (always suggest help) and NEVER (never suggest help) as well as the adaptive policy THRESH (suggest help if its quality exceeds a certain threshold). The task given to the users was to edit text with a Dvorak[3] keyboard.

They got the result that 3 out of 4 users actually preferred ALWAYS to their adaptive policy MEU. However, the system inferred that all 4 users were of the needy (TN = true) and dependent (TI = 1) type which can explain why they preferred ALWAYS over their adaptive policy.

Task Recognition

In the previous section we focused on adapting the user interface based on the user's current needs and attitudes. These user features naturally change over time because the user's tasks change over time. To examine this aspect in more detail we review a paper that focuses on predicting the user's current activity (task).

A group at the Oregon State University has developed the *TaskTracer* (Dragunov *et al.* 2005) system which aims to help multi-tasking users manage their resources. They do this by associating all accessed files, folder and emails of a user with user-defined activities. *FolderPredictor* (Bao, Herlocker, & Dietterich 2006) is an application that builds on top of the TaskTracer system and modifies the open/save dialogue to give the user the most probable folders for the current task (see figure 8).

The initial TaskTracer system relied on the users to manually switch the task they are currently working on. Because of this inconvenience they developed *TaskPredictor* (Shen *et al.* 2006) as a part of the TaskTracer system, which tries to predict the user's current activity based on the observation of accessed objects (e.g. windows, files). We are focusing on TaskPredictor in the following.

[2]Maximum expected utility

[3]A different keyboard layout than QWERTY

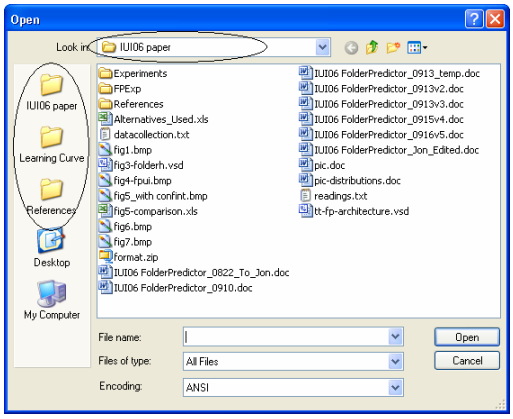

Figure 8: FolderPredictor, an application of TaskTracer.

TaskPredictor. The TaskPredictor consists of two parts, namely *Taskpredictor.WDS* and *TaskPredictor.email*. Taskpredictor.WDS aims to predict the task based on events, called *Window-Document-Segments (WDS)*, whereas TaskPredictor.email predicts the task based on incoming email messages. We will concentrate at TaskPredictor.WDS in the following, but TaskPredictor.email is similar.

"A WDS consists of a maximal contiguous segment of time in which a particular window has focus and the name of the document in that window does not change." (Shen *et al.* 2006)

TaskPredictor.WDS. TaskPredictor.WDS extracts the information from the WDS (window title, filename, path, and website url if applicable). It then segments the information into a set of words and adds them to a global set of unique words that have been collected from all events so far. For each event, it keeps the feature set of the event's unique words (see figure 9). TaskPredictor.email is very similar, it just uses the properties of the incoming emails instead (sender, recipients and subject).

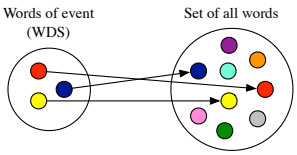

Figure 9: Feature set

Machine Learning Methods. In order to make the predictions, they employ three machine learning methods: *Classification thresholds*, *Mutual information feature selection* and a *Hybrid Naive Bayes / Support Vector Machine (SVM) classifier*.

First, they apply feature selection to reduce the amount of features that will be taken into account for the prediction.

The intention is to improve the accuracy of the classifier by reducing its complexity and additionally gaining speed by doing that.

Classification thresholds are used to decide whether the system should make a prediction or not, based on the classification confidence.

A Naive Bayes classifier is used to compute the classification confidence. It estimates the probability density on the neighbourhood of the current event. If there have been similar events before, it is more likely that the current event refers to a specific task. It is also fast to compute because it assumes that all features (here words) are independent.

The actual prediction of the user's task is done by a Support Vector Machine (SVM). A SVM is a linear classifier that separates objects into two classes such that the classborder has the most possible margin to all objects. It is therefore called a large margin classifier and has been shown suitable for text prediction (Shen *et al.* 2006).

Classification thresholds. Let x be a vector of features extracted from a WDS and y be a task. Then the probability of x is

$$P(x) = \sum_y P(x|y)P(y)$$

which is calculated by the Naive Bayes classifier and estimates the probability density on the neighbourhood of x. Then they use a threshold θ and only make a prediction if $P(x) > \theta$, i.e. the prediction confidence exceeds the threshold.

Feature Selection. A stopword list is used to get rid of very common words (like "open", "to") because they would otherwise appear in a lot of events and make the prediction less accurate. They also apply stemming of words before they store them in the feature set (e.g. jumping → jump). Finally *mutual information* is used to choose the 200 features with the largest (individual) predictive power. "Intuitively, mutual information measures the information that X and Y share: it measures how much knowing one of these variables reduces our uncertainty about the other." (Wikipedia 2006)

Naive Bayes + SVM classifier. As mentioned previously, Naive Bayes is cheap to compute to estimate the probability of an event. Support Vector Machines are supposed to have higher predictive power in general.

The Naive Bayes classifier is used to estimate the probability $P(x)$ of the current event x. Then the actual prediction of the most probable task is made with the SVM, but only if $P(x)$ exceeds the given threshold θ. If $P(x) \leq \theta$, no prediction is made because the system in not confident enough to predict the current activity (according to θ).

Results. The results are measured in terms of *coverage* and *precision*. Coverage denotes the fraction of cases in which a prediction was made, in other words the system was confident enough in its prediction. Precision is the probability that the prediction being made is correct.

The effect of the classification threshold θ: A low θ causes high coverage with a low precision. High θ results in low coverage with high precision on the opposite. The overall goal is to have high precision, even at low coverage. This is because the system doesn't want to interrupt the user if it is not very confident in the prediction that can be made. So they fixed the precision to 80% and found out that they could get a coverage of 10% and 20% (they only had two subjects). They even got better results for TaskPredictor.email, they could get 92% precision with 66% coverage.

They got slightly better results with their hybrid classifier than with either single Naive Bayes or SVM as can be seen in figure 10.

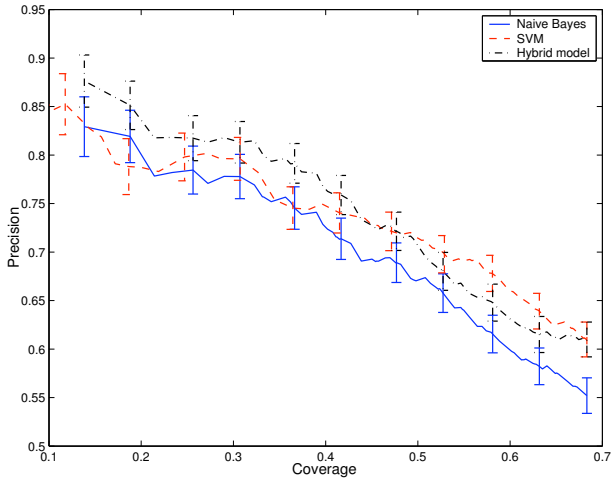

Figure 10: Results for TaskPredictor.WDS (Shen *et al.* 2006)

Summary and Future Directions

We have discussed how to use user modelling for implementing intelligent user interfaces. Decision networks can be used to decide if a user interface should assist users, but it has to be taken into account that this is a interruption which has a cost associated with it. In most cases the cost is even higher if the suggestion is poor or even wrong. User models can be rather complex, but simply reducing the complexity will in most cases lead to a tool with limited intelligence which has no use or is even annoying for the user.

We have seen the approach of (Hui & Boutilier 2006), which aimed to infer the user's features explicitly to decide when to offer help to the user. They developed a general user model which can be used in different domains and demonstrated its feasibility for the task of text-editing where the user's goal is typing words. They are currently applying their model to a different domain with changing user goals, so they have to alter the prediction model to take that into account.

(Shen *et al.* 2006) focused on the classification of user activities. They did this in order to support them fulfilling their tasks, e.g. by modifying the open/save dialogue for

applications (see Figure 8). They employed several machine learning methods, especially a hybrid Naive Bayes / SVM classifier, to increase the accuracy of the prediction being made.

We think that both components, knowing the user's features (including emotional state and personality) and knowing what the user is currently doing, are necessary to improve the quality of modern user interfaces.

To improve intelligent user interfaces the range of observations should be broadened to enable a more precise prediction. Intelligent systems should be tested with users to find out if the automatic adaptation is accepted by users and does not interrupt too much. Although user interfaces are very complex today and automatic adaption is helpful, users also want to be able to predict how the interface behaves. These are competing goals and it is a very difficult to find the right trade-off.

Acknowledgement

We thank Bowen Hui for her helpful explanations and comments about her paper and current work.

References

Bao, X.; Herlocker, J. L.; and Dietterich, T. G. 2006. Fewer clicks and less frustration: reducing the cost of reaching the right folder. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, 178–185. New York, NY, USA: ACM Press.

Dragunov, A. N.; Dietterich, T. G.; Johnsrude, K.; McLaughlin, M.; Li, L.; and Herlocker, J. L. 2005. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, 75–82. New York, NY, USA: ACM Press.

Horvitz, E.; Breese, J.; Heckerman, D.; Hovel, D.; and Rommelse, K. 1998. The Lumière project: Bayesian user modeling for inferring the goals and needs of software users. In *In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 256–265. Madison, WI: Morgan Kaufman.

Hui, B., and Boutilier, C. 2006. Who's asking for help?: a bayesian approach to intelligent assistance. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, 186–193. New York, NY, USA: ACM Press.

McGrenere, J.; Baecker, R. M.; and Booth, K. S. 2002. An evaluation of a multiple interface design solution for bloated software. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, 164–170. New York, NY, USA: ACM Press.

Shen, J.; Li, L.; Dietterich, T. G.; and Herlocker, J. L. 2006. A hybrid learning system for recognizing user tasks from desktop activities and email messages. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, 86–92. New York, NY, USA: ACM Press.

Wikipedia. 2006. Mutual information — wikipedia, the free encyclopedia. [Online; accessed 3-December-2006].

Zhang, N. L., and Poole, D. 1996. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research* 5:301–328.