Predicting Future User Actions by Observing Unmodified Applications

Peter Gorniak and **David Poole**
Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada
pgorniak@cs.ubc.ca, poole@cs.ubc.ca

Abstract

Intelligent user interfaces often rely on modified applications and detailed application models. Such modifications and models are expensive to build and maintain. We propose to automatically model the use of unmodified applications to solve this problem. We observe a user's interactions with the application's interface and from these observations infer a state space which the user navigates and the stochastic policy he or she follows. ONISI, the algorithm presented here, builds this state space implicitly and on-line, and uses it to predict future user actions. Trials with real users show that this algorithm predicts the next user action significantly better than another current algorithm.

Content Areas: plan recognition, human computer interaction, automated modeling, software agents

Introduction

Artificial Intelligence supplies a vast set of tools to be applied to the design of intelligent user interfaces. While our previous research (Gorniak 1998) as well as countless other projects sample indulgently from this set and often produce quite impressive results in their own environments (for example, see (Horvitz, Breese, Heckerman, Hovel and Rommelse 1998) and (Albrecht, Zukerman, Nicholson and Bud 1997),) there emerge some new challenges when one attempts to apply these results to a new application.

1. The research results often do not transfer easily to a new application.
2. The actual implementation used in the research relies upon a modified application. This modification is usually non-trivial, time-consuming to repeat and increases application complexity.
3. Researchers work from various, often hand-crafted application models. In addition to the application building work in 2, an application designer needs to specify such a model. This process is often not straightforward and may rely on empirical data from user trials. An application designer's primary task does not include designing such a model, and thus the task seems an added difficulty to him or her. Also, the model needs to be updated and will tend to lag behind the application during maintenance.

We are currently addressing these problems by investigating how much knowledge can be extracted from a user's interaction with an application without any prior information about the application's purpose or structure and without any modifications to the application (somewhat in the spirit of (Lieberman 1998).) We hypothesize that enough knowledge can be extracted to yield a detailed model of the application and its user. This model can then serve both as a knowledge source for other algorithms as well as provide a context under which to unite methods.

Other application independent user models build no application model at all, and thus do not provide any automatic analysis of the application. We show that they perform worse in cases where such application knowledge boosts performance, such as future action prediction (Davison and Hirsh 1998). Others stop early on in their analysis and subsequently rely on application specific knowledge (Encarnacao 1997). Our approach is similar to the web precaching work by Zukerman, Albrecht and Nicholson (1999), but they do not perform any kind of history analysis for state identification, except for simple frequency counts. We show here that frequency counts generally do not capture the user's current state as well as other possible identifiers.

In this paper, we employ a user modeling strategy to predict future user actions. It was our hope in choosing this domain that our extensive use of structured observations and history analysis would increase our prediction accuracy as compared to simpler approaches. Also, knowledge about future user actions can be useful to achieve various goals in many applications, thus providing an ideal target for a general application independent approach like ours. Specifically, predicting the user's next action allows application designers to automate tasks for the user, to initiate time-intensive tasks before the user requests them and to identify a user by comparing his or her real actions with the predicted ones. For example, Davison and Hirsh (1998) use next action prediction for UNIX command line completion. Current web browsers support a similar feature for URL completion. Debevc, Meyer, Donlagic and Svecko (1996) developed an adaptive toolbar that adjusts its displayed collection of tools based upon user behaviour. A next action prediction algorithm performs exactly the kind of analysis needed to make decisions in such adaptive interfaces. Zukerman et al. (1999) pre-cache web pages for users by predicting

their next request. Lane (1999) uses Hidden Markov Models to detect security breaches by comparing a user's current behaviour to the HMM derived from their past behaviour. We believe the same goal can be achieved by comparing the predictions made by a next action prediction algorithm to the real user actions. Finally, knowledge about future user actions is valuable to intelligent help systems. These systems can use such information to identify user plans and help the user to achieve his or her current goals. We expand further on how our approach to user modeling can be used to support other intelligent interface components in (Gorniak and Poole 2000).

Let us view the user as an agent. Our assumption is that we can and have observed this agent acting in an environment, namely using an application. Artificial Intelligence concerns itself with agents acting in environments and worries about what decisions such agents should make. A common approach to such a problem consists of phrasing it in terms of states and actions between states and coming up with a policy that, perhaps stochastically, dictates which actions to take in which states (Boutilier, Dean and Hanks 1999). We are faced with the opposite problem: we see an agent acting in an environment and want to model the agent's decision process. We assume that the agent acts according to a policy. Each action is the result of some (possibly stochastic) function of what the agent observes and the agent's belief state. Our goal is to determine this policy and the state space to which it applies.

The following section describes our approach in detail. Our algorithm performs an on-line implicit state identification (ONISI). That is, it assigns probabilities to all possible actions in the currently observed interface state. It arrives at these probabilities by estimating how much support the observed interaction history contains given the history recorded immediately prior to entering the current state. To do so, it employs a k nearest neighbours scheme that uses sequence match length as a metric.

The section on implementation describes the Java implementation of the work presented here. This implementation works as a wrapper to existing Java applications and is able to record their interfaces states as well as user actions without modifications to the original application. The results section compares our algorithm, ONISI, to IPAM (Davison and Hirsh 1998), another next action prediction algorithm in user trials and shows that our algorithm performs significantly better by exploiting behavioural patterns found in the recorded user history. Finally, we conclude by discussing what the research presented here reveals about the issues involved in analysing user history and point to future research directions.

State Identification Algorithms

A state for the user consists of a combination of the user's internal state and the application's interface state. At a given time, the user chooses an action from a probability distribution based upon the current state. We attempt to determine the policy the user is employing from our observation of the user's interaction history. To do so, we hypothesize internal states of good predictive power in addition the observed interface states by searching the observed interaction history for behavioural patterns. These implicitly identified states refine the frequently very coarse observations of application interface states and let us predict more accurately what action the user will choose next. In the following, we call the currently observed interface state s and the possible next action currently under consideration a.

In the context of this work the only information to identify the current state is the observed interaction history. There are several choices we must make as to how to mine this information. We need to choose

1. A type of pattern to extract from the interaction history that can be matched against recently occurring actions,
2. A method to summarize the occurrence of a pattern in history,
3. A function that ranks the currently possible actions according to the summaries of applicable patterns.

IPAM (Davison and Hirsh 1998) exemplifies one possible set of choices. They choose pairs of actions occurring in sequence as a pattern and summarize them by increasing the probability of those that occur and decreasing the probabilities of all others. Finally, they rank the currently possible actions by considering all the pairs that start with the action that occurred one timestep ago and selecting the one of highest estimated probability to predict the next action. These choices make an implicit Markov assumption, namely that the last action together with the current summary provided by the probability estimates contain enough information to predict the next state. Looking at real user interaction traces we found that often users enter modes that can be easily identified by examining the behavioural patterns they engage in, but that these patterns span more than two actions in a row. For example, in the search algorithm application we investigated users would either step through a problem or simply look at an algorithm's result. Both of these modes can be identified and the actions that occur in them predicted by looking several steps back. IPAM fails to do so, and cannot make a significant fraction of predictions in this case.

To remediate this problem, we decided to automatically vary the length of patterns we identify. Indeed, we deem patterns more important to the state identification process if they are longer, building on the idea that longer patterns capture the current state better. Overall, we choose our patterns to be the longest sequences in history that match the immediate history. Given a small integer k, at time t in state s we summarize the sequences that predict action a by computing $l_t(s, a)$: the average of the lengths of the k longest sequences that end with action a in state s and match the history sequence immediately prior to time t. We rank currently possible actions according to this summary value for each action (i.e. according to the average of the lengths of patterns the action completes.) Due to our decision to view match length as the important criterion, we only consider k maximum length sequences so as to avoid making the average less distinctive by considering more. In short, rather than estimating the probability of the next action given the

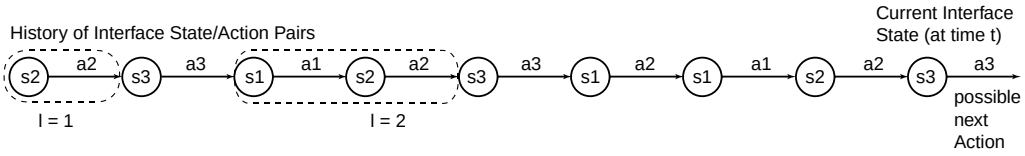

Figure 1: Example for On-Line Implicit State Identification

previous action like IPAM does, we estimate the probability of the next action given as much as we deem significant of the history that occurred immediately before this timestep. We determine significance by whether this sequence occurs previously in the observed history. This approach leans on a similar one used to identify hidden states during reinforcement learning (McCallum 1996). It is also akin to automation approaches that identify repeated patterns in user history (Ruvini and Dory 2000), but has the more general goal of modeling the use of an unmodified application.

Our measure of history support assigns no importance to how frequently actions occur, but only to how long the patterns are that they complete. However, action frequencies do encode a type of history support for an action that may augment the type of support we capture using match length. Specifically, a certain action may not be a part of long patterns, but its occurrence may be strongly correlated with that of another action. To account for this possibility, we reintroduce a simple frequency measure ($f(s, a)$), namely how many times an action has been taken from the current state. This term measures only state-action correlation and not action-action correlation like IPAM does, but it is less expensive to compute than IPAM's probability estimate. We experimentally traded off between the match length measure normalized across all possible actions,

$$\frac{l_t(s,a)}{\sum_i l_t(s,a_i)}$$

and the frequency measure, also normalized,

$$\frac{f(s,a)}{\sum_i f(s,a_i)}$$

using a parameter $0 \leq \alpha \leq 1$ to see when each would be useful. Figure 2 summarizes the ONISI k-nearest neighbours algorithm.

When normalized across all possible actions from the current state, $R(s, a)$ can be interpreted as the current probability of each action.

Figure 1 shows an example step of ONISI on a short history sequence. The application is currently observed to be in state $s3$ and the algorithm is ranking the possible action $a3$ from that state with $k = 3$. As shown, it finds $\{3, 1, 0\}$ as the set of maximum length sequences matching the immediate history, and thus calculates

$$l_t(s3, a3) = \frac{0+1+2}{3} = 1.$$

Assuming that all actions provide a sum $\sum_i l_t(s, a_i) = 5$, that action $a3$ has occurred 50 times in interface state $s3$,

Given the currently observed interface state s, the action a currently under consideration, a small integer k to indicate how many pattern matches to consider and a real value $0 \leq \alpha \leq 1$ to indicate how to weigh off between the match length measure and the frequency measure, where $\alpha = 1$ uses only the match length measure and $\alpha = 0$ only the frequency measure,

1. Compare the immediate history starting at t with the state-action pair (s, a) and running backwards through all the recorded history. Find the k longest sequences in the recorded history that match the immediate history (they can be length 0).
2. Average the length of these sequences and call this average $l_t(s, a)$.
3. Count the number of times a has occurred in s and call this $f(s, a)$.
4. Return ranking

$$R_t(s,a) = \alpha \frac{l_t(s,a)}{\sum_i l_t(s,a_i)} + (1-\alpha)\frac{f(s,a)}{\sum_i f(s,a_i)}$$

where the sums run over all possible actions from s.

Figure 2: The ONISI k-nearest neighbours algorithm

and that $s3$ has been visited 100 times overall, ONISI run with $\alpha = 0.9$ finally assigns a rank of

$$R_t(s3, a3) = 0.9\frac{1}{5} + 0.1\frac{50}{100} = 0.18 + 0.05 = 0.23$$

to action $a3$ in observed interface state $s3$ at time t.

Implementation

Java's reflective capabilities and dynamic loading strategy make the language a prime candidate for an application independent approach (JDK 1998). It allows not only inspection of a structure of known visual components, but it can also inspect unknown components for state information. Java and JavaBeans introduced standard naming conventions for object methods. For example, *isVisible()* returns the visibility status of visual components, whereas *getEnabled()* returns whether they are currently useable. Components derived from standard Abstract Window Toolkit components inherit these methods automatically, and other components should define them. Java's Reflection mechanism, on the other hand, allows one to check whether a given object includes one of these state-revealing methods, and lets one call this

method without knowing the object's class. Finally, Java's dynamic loading of classes rids the developer of needing to link with or even know about classes that will be present at runtime. Using these tools, one can establish the user interface state of an application built using Java at runtime by dynamically linking into its code, examining the methods available in its objects and calling the methods relevant to the interface state. This process requires no modification of the targeted application at all.

The system used for the experiments presented below runs as a wrapper to a Java application. Before it starts the application, it hooks itself into the application's event queue and thus sees all event activity within the Java Abstract Window Toolkit and components derived from it. It intercepts each such event that it considers an action (such as a button being pressed or a window closed) and records the observed state of the application's interface before and after the event occurs. In this way, this system establishes a state space of interface observations as a person uses the application and records a history consisting of actions and visited states at the same time.

The applications[1] under consideration here are educational AI applications. They were written to help undergraduate university students learn concepts in Artificial Intelligence. One application familiarizes the student with search problems and algorithms, the second deals with constraint satisfaction problems and the third demonstrates backpropagation neural network learning. In each, the student has the option to either load an example problem or to create his or her own problem by drawing a graph. He or she can then switch to a problem solution mode and step through the various algorithms at different levels of detail. The students used these applications to solve homework problems for an introductory AI course they were taking. Most of the assignment questions referred to a supplied example problem, so the students tended to explore the problem creation facilities of the applications less than their solving functionality. The following discussion and results focus mainly on the application for search algorithms.

Results

ONISI depends on two parameters: k and α. Figure 3 shows a graph of ONISI's performance over a range of values for these parameters, measured in percentage of actions predicted correctly for the search algorithm application (the application that yielded the largest dataset of about 2200 user actions.) The other two applications show similar trends, but due to the smaller dataset sizes they are less distinct. The graph continues to level out for larger values of k and performance continues to get worse for smaller values of α. Note that figure 3 is greatly magnified, and thus detailed features of the graph are likely not significant and should be ignored.

Concerning k, the graph shows the behaviour one would hope for in a nearest neighbour approach: small values of k show the same performance as larger ones, and thus suffice to capture history support accurately. As for α, a setting that

[1]The applications can be found at *http://www.cs.ubc.ca/labs/lci/CIspace/*.

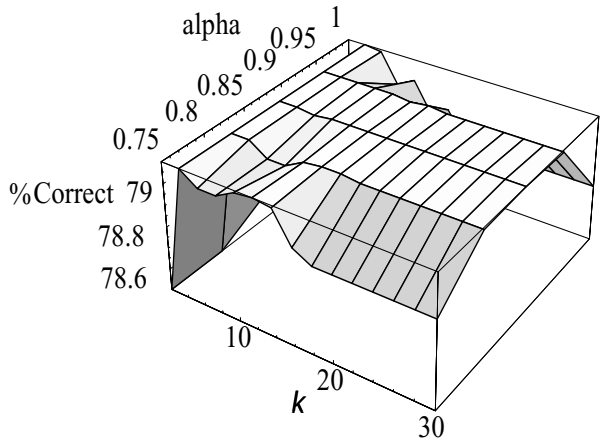

Figure 3: Performance at various parameter settings

assigns almost all importance to the match length measure yields the best performance, but if the frequency measure is ignored ($\alpha = 1.0$,) performance degrades. Upon inspection, the cases that are successfully predicted at $\alpha = 0.9$, but not at $\alpha = 1.0$ are indeed those where there are k maximal (but short) length matches in history for several actions and the default random choice between them that is used at $\alpha = 1.0$ performs worse than using the frequency measure to distinguish between them by setting $\alpha = 0.9$. All the following experiments were run with $k = 5$ and $\alpha = 0.9$.

We compared the implicit version of our state space approximation to IPAM which its authors in turn compare to a number of others (Davison and Hirsh 1998). IPAM estimates $P(a_{t+1}|a_t = x)$, i.e. the probability of the next action given the previous action. To do so, they build a table with all possible actions as rows and as columns and update its entries to be their desired probabilities during runtime. Specifically, they decrease all action pair probabilities in the row of the current action by a factor of $0 < \alpha < 1$ (note that this α is different from the α ONISI uses to trade of between frequency and length measure) and boost the probability of the action pair that just occurred by adding $1 - \alpha$, thus ensuring that the probabilities in a row continue to add up to 1.

We found IPAM to perform best on our data with a value of $\alpha = 0.9$, and all results here occur with that setting. The graphical user interface setting under investigation here differs somewhat from the UNIX command line prediction IPAM was designed for, and so does the state identification framework. To level the playing field, we let IPAM consider observed action-state pairs of the interface to be individual actions for its predictions. Taking state into consideration in this way produces better action distinctions, and IPAM's performance increases slightly (by ca. 2% for the search application.)

Figure 4 shows the percentage of actions predicted correctly by ONISI and IPAM for the search, neural network and constraint satisfaction applications. The numbers in parentheses indicate the number of actions recorded for each

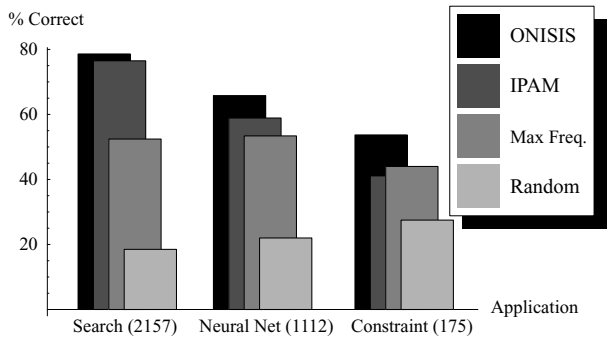

Figure 4: Performance of ONISI vs. IPAM

application. As baselines, the chart also indicates the percentage of actions predicted correctly using just the maximum action frequency in a state (ONISI with $\alpha = 0$) and by picking a random action amongst all that have been observed in a state. In each case the algorithms ran in a realistic setting with no access to actions that lie in the future. There are few users who performed enough actions to allow for meaningful analysis on a per-user basis, so the data were treated as if produced by a single user in successive sessions. We believe that the applications are simple enough and the user's goals and experience levels similar enough to validate this simplification. The differences in the percentages of actions that can be predicted for each application stem partially from the differing number of actions recorded for each, but also from the nature of the application. First, the random prediction baseline shows that as a tendency the number of possible actions for a given interface states increases with the number of total actions we record from users. Second, the frequency measure demonstrates that slightly less than half the actions in any of the applications can simply be predicted from the frequency distribution recorded in an interface state. Third, It is expected that with less data to predict from all prediction performances decrease, except that of the random predictions. As expected, ONISI degrades more gracefully than IPAM, because IPAM requires some time for its predicted probabilities to converge.

As for the difference in prediction performance of ONISI and IPAM as compared to the frequency measure in the search and neural network applications: the recorded history for the search application includes long runs of the students stepping through a problem. These runs consist entirely of one action, and are easy to predict for either algorithm, making the overall prediction percentages of ONISI and IPAM high and close in value. Neither the neural net nor the constraint application show this type of student behaviour in the histories. To document this fact we recorded the average maximum match length ONISI detects when finding the k nearest neighbours. For the search application, this average lies at a length of 14.2 steps, whereas for the Neural Net application it is 4.6 steps, indicating that indeed long sequence matches are abundant in the search application.

Overall, ONISI performs better for each application with statistical significance at greater than 95% confidence as given by a chi-square test using a directional hypothesis for each of the applications. Why does ONISI perform better? Some clear patterns emerge when looking at the specific actions that it predicts but IPAM (and, we suspect, other approaches that work from a very short, fixed length interaction history) fail to predict:

- While the IPAM probability estimates take some time to converge for a new user, ONISI is able to identify a distinct behavioural pattern if it has only occurred once before.
- ONISI successfully identifies state information hidden in the user's history beyond the last action. For example, there are two modes in which users seem to learn about search algorithms with the search application. In one mode, a user will switch to a new search algorithm and step through the problem by hand to learn how the algorithm works.

State	**Action**
Stepping	ActionEvent on Step
Goal Node Reached after Stepping	ActionEvent on Ok
Stepping	ActionEvent on Reset Search
Problem Solution Mode	ItemEvent on Breadth First Search
Problem Solution Mode	ActionEvent on Step

Table 1: Typical Search Action Sequences

In the other mode, as exemplified in Table 1, a user wants to learn about the result of different search algorithms, rather than the process, and immediately displays this result after switching to a new search algorithm. The problem for IPAM consists of the fact that the last action, namely the switch to a different search algorithm, does not encode the mode in which the user is currently using the application. In fact, often even the last few actions are not sufficient to identify this mode, because the user has likely just finished another search problem and clicked through a sequence of result dialogues - a sequence that is the same for either mode. ONISI, however, easily distinguishes these modes, because it finds some nearest neighbours that match the whole history sequence in the Table 1 and further actions. These are of greater length than sequences that only match backwards to where the goal node was reached, but contain the actions to immediately show a search result before that point.

Even more interesting than the fact that ONISI performs better than IPAM is that they predict different actions correctly. For the search application, there are 160 cases (ca. 7.5%) in which one of the algorithms predicts correctly and the other does not. This means that one does not replace

the other, and in the concluding section below we argue that there is a systematic reason for this that can be exploited in further research.

Conclusion and Future Work

We have successfully designed and implemented a system that builds a model of an application, its graphical user interface and its user without requiring any modification of the application or any interaction with the user. We showed this approach can be used to predict future user actions by establishing the state space the user navigates together with the policy he or she uses to decide upon actions. To do so, we establish interface states of the application to build a coarse version of this state space and subsequently refine it. Our refinement algorithm identifies the k nearest neighbours to the immediate history and a possible next action and from their length estimates the action's likelihood, thus implicitly identifying the current state. This prediction method works better than other known future action prediction algorithms, but differs from them in which actions it can predict successfully.

Throughout the paper we identify the main differences between our approach and IPAM. These differences point to some of the main issues to be addressed:

1. ONISI and IPAM predict based on very different features of the recorded history. ONISI finds matching behavioural patters, whereas IPAM collects statistics about pairs of actions that occur in sequence. ONISI performs better than IPAM for the data in question in this paper, but the choice of predictor seems somewhat arbitrary.
2. ONISI and IPAM are able to predict different actions. To us, this indicates that neither of them is the ideal solution to the next action prediction problem.
3. An added simple frequency measure enhances our match length based implicit state identification.

These issues all point to the fact that the choice of patterns to mine from history and the measure to use in interpreting them dictate which actions a given algorithm can successfully predict. Ideally, the lessons learned from comparing ONISI to IPAM will let us design an algorithm that can trade off between the possible choices and draw from the predictive success of both approaches.

Finally, we are currently working on OFESI, an explicit, off-line version of ONISI (Gorniak and Poole 2000). That algorithm has a different goal from ONISI in that it should introduce new states when they explain a large fraction of actions in the overall application usage. ONISI, on the other hand, predicts the next action successfully at a given point in time even if that action occurs extremely rarely. Our results show that the explicit state graph inferred by OFESI captures the application and its usage well and can serve as a tool for application designers to analyse the application and to augment it with other intelligent interface components. Also, there are strong relationships between building such a stochastic dynamic model and deriving states for Hidden Markov Models, opening up a whole new realm of applications in the general field of sequence modeling (Rabiner 1989, Seymore, McCallum and Rosenfeld 1999).

References

Albrecht, D. W., Zukerman, I., Nicholson, A. E. and Bud, A.: 1997, Towards a bayesian model for keyhole plan recognition in large domains, *User Modeling: Proceedings of the Sixth International Conference, UM97*.

Boutilier, C., Dean, T. and Hanks, S.: 1999, Decision-theoretic planning: Structural assumptions and computational leverage, *Journal of AI Research* **11**, 1–94.

Davison, B. D. and Hirsh, H.: 1998, Predicting sequences of user actions, *Technical report*, Rutgers, The State University of New York.

Debevc, M., Meyer, B., Donlagic, D. and Svecko, R.: 1996, Design and evaluation of an adaptive icon toolbar, *User Modeling and User-Adapted Interaction* **6**(1), 1–21.

Encarnacao, L.: 1997, *Concept and Realization of intelligent user support in interactive graphics applications*, PhD thesis, Eberhard-Karls-Universität Tübingen, Fakultät für Informatik.

Gorniak, P. J.: 1998, Sorting email messages by topic. Project Report.

Gorniak, P. J. and Poole, D. L.: 2000, Building a stochastic dynamic model of application use. Forthcoming.

Horvitz, E., Breese, J., Heckerman, D., Hovel, D. and Rommelse, K.: 1998, The lumiere project: Bayesian user modeling for inferring the goals and needs of software users, *Uncertainty in Artifical Intelligence, Proceedings of the Fourteenth Conference*.

JDK: 1998, *Java Development Kit Documentation*. **URL:** *http://java.sun.com/products/jdk/1.1/docs/index.html*

Lane, T.: 1999, Hidden markov models for human/computer interface modeling, *Proceedings of the IJCAI-99 Workshop on Learning about Users*, pp. 35–44.

Lieberman, H.: 1998, Integrating user interface agents with conventional applications, *Proceedings of the International Conference on Intelligent User Interfaces, San Francisco*.

McCallum, A. R.: 1996, Instance-based state identification for reinforcement learning, *Technical report*, University of Rochester.

Rabiner, L.: 1989, A tutorial on hidden markov models and selected applications in speech recognition, *Proceedings of the IEEE*, Vol. 77(2).

Ruvini, J.-D. and Dory, C.: 2000, Ape: Learning user's habits to automate repetitive tasks, *Proceedings of the International Conference on Intelligent User Interfaces*.

Seymore, K., McCallum, A. R. and Rosenfeld, R.: 1999, Learning hidden markov model structure for information extraction, *AAAI-99 Workshop on Machine Learning for Information Extraction*.

Zukerman, I., Albrecht, D. and Nicholson, A.: 1999, Predicting users' requests on the www, *User Modeling: Proceedings of the 7th International Conference, UM99*.