ARTÍCULO

A User Profiling Architecture for Textual-Based Agents

Daniela Godoy[1] and Analia Amandi

ISISTAN Research Institute, UNICEN University
Campus Universitario, Paraje Arroyo Seco (B7001BBO) Tandil, Bs. As., Argentina
[1]Also CONICET
{dgodoy; amandi}@exa.unicen.edu.ar

Keywords: intelligent agents, user profiling, personalization.

Several intelligent agents have been developed in the last decade to help users with the vast amount of information available in the World Wide Web (WWW). Despite the efficiency of these agents depend on the knowledge they have about users, which is contained into user profiles, there are diverse considerations about what a profile should contain and how to construct it. Due to this fact, developers have to face the problem of, not only specifying the user profile content each time, but also its acquisition and adaptation to change in user interests. In this work we present an architecture which prescribes these aspects of the user profiling task. The goal of this architecture is to guide developers in the construction of agents involved with textual-based tasks. The results obtained from its application in a search agent, called *PersonalSearcher*, are also reported in this work.

A User Profiling Architecture for Textual-Based Agents

Daniela Godoy*, Analía Amandi

ISISTAN Research Institute, UNICEN University
Campus Universitario, Paraje Arroyo Seco (B7001BBO)
Tandil, Bs. As., Argentina
*Also CONICET
{dgodoy; amandi}@exa.unicen.edu.ar

Abstract

Several intelligent agents have been developed in the last decade to help users with the vast amount of information available in the World Wide Web (WWW). Despite the efficiency of these agents depend on the knowledge they have about users, which is contained into user profiles, there are diverse considerations about what a profile should contain and how to construct it. Due to this fact, developers have to face the problem of, not only specifying the user profile content each time, but also its acquisition and adaptation to change in user interests. In this work we present an architecture which prescribes these aspects of the user profiling task. The goal of this architecture is to guide developers in the construction of agents involved with textual-based tasks. The results obtained from its application in a search agent, called *Personal Searcher*, are also reported in this work.

Keywords: intelligent agents, user profiling, personalization

1 Introduction

In the last decade personal agents have emerged as a solution to deal with the increasing volume of information available in the World Wide Web (WWW). These agents are intelligent assistants that make different tasks on behalf of users to find, filter and access to large amounts of information, and present a reduced and potentially relevant part of this information to their users. They act as human assistants, collaborating with the user and becoming more efficient as they learn about user interests, habits and preferences. Agents developed in this area have addressed many tasks, such as filtering, searching and browsing the Web [9, 6, 7].

Personal agents rely on having some knowledge about users into user profiles i.e. models of users preferences and interests by which agents can assist user activities. Despite user profiling has become a key area in agent development, since agents effectiveness depend on profiles precision, there is not a consensus about what a profile consists of, how profiles are constructed and how they could be used to assist users [13].

For the majority of the agents performing textual-based tasks currently in existence, the information contained within profiles is merely the set of interests a user has and, in some cases, also his dislikes. These interests regarding different topics are represented for each agent according to its own approach at different detail levels. Consequently, profile contents range from just a few keywords provided by the user to multiple topics of interest obtained by observation and defined by keywords.

The diversity of profile contents found in these agents is the result of the application of different alternatives

to the user profiling task. Existent approaches can be broadly classified as originated on either information retrieval or machine learning techniques. Approaches in both groups have shown different disadvantages at the moment of being applied to the user profiling task in textual contexts. In approaches of the first group, keywords belonging to interesting documents are analyzed in isolation from the rest of the user reading experience during which they were captured (e.g. user tasks context, etc.), thus losing contextual information about user interests. In contrast, those approaches from the second group can easily capture contextual information but their application to textual data is not straightforward. The main problem related to the usage of machine learning algorithms to acquire user models is the computational complexity arising with the amount of information to be treated. These particular aspects regarding each approach need to be solved by agent developers in order to efficiently build accurate user models.

In this context, developers have to face the problem of defining for each agent, not only what they should know about a user, but also how to obtain this knowledge and how to apply it later to provide personalized assistance. Moreover, they also need both to identify which parts of a user profile content will be affected as the user behavior changes over time and to specify how this adaption will take place. Without any support to carry out these tasks, agent development becomes a difficult and time consuming task, besides restricting agents capabilities to the possibilities provided by the adopted approach.

In this work we propose an architecture for user profiling to be applied in the development of textual-based agents. The main objective of this architecture is to guide developers in the construction of agents assisting users dealing with texts on the Web. It could be applied in the construction of browsing assistants, search agents, etc.

The user profiling architecture prescribes the implementation of necessary tasks in the user profiling process, including the user profile content acquisition, its adaptation to changes and its possible applications to provide a personalized assistance. The capabilities of agents developed under this architecture are: answering about the relevance of a given information to the user profile, pro-actively suggesting information about certain topics in proper time and sharing information with other agents to establish common interests of a user community.

In order to evaluate the architecture we implement a search agent, named *PersonalSearcher*, based on it. This agent assists users to find interesting documents on the Web by carrying out a parallel search in the most popular Web search engines, filtering results and presenting a reduced number of documents with high probability of being relevant to the user.

This work is organized as follows. Section 2 delineates the architecture for user profiling. The main components of this architecture are explained in Section 3. The *PersonalSearcher* agent is described in Section 4. Section 5 discusses related works. Finally, conclusions are presented in Section 6.

2 User Profiling Architecture

An architecture of a specific system is a collection of computational components or simply components, with a description of the interactions between these components, the connectors. Figure 1 shows components and interactions of the user profiling architecture and how it is used in the context of an agent performing a textual-based task. On the left, it is shown how the architecture updates the content of a user profile. Dotted lines in the figure indicate the communication between the architectural components and the different parts of the user profile content. On the right, the architecture interacts with components belonging to the agent itself which, in turn, interacts with the environment, e.g. by receiving events from the user, performing actions to assist him and communicating with other agents.

Agents observe events on the environment, they understand what they perceive by reasoning about it, and execute actions depending on their current mental state. Since *observation*, *action* and *communication* depend on the application being developed, they can not be prescribed by the user profiling architecture. We do not force the developer to use a specific design of these components. As a result, our architecture can be combined with any general agent architecture such as BDI [11] or InterRap [8].

The agent *observation* component is required to collect experiences about interesting documents read

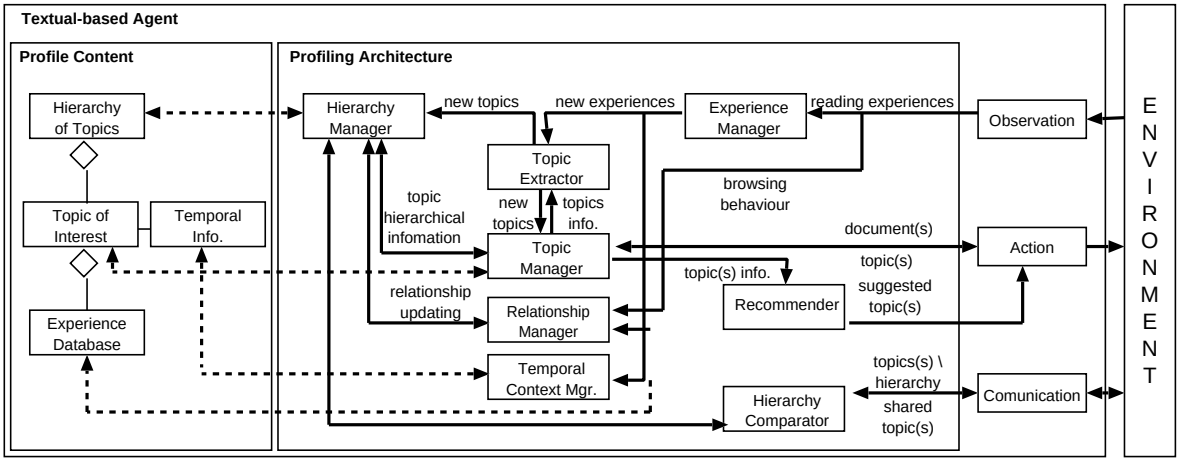

Figure 1: User Profiling Architecture

by the user. Regardless of how it collects these experiences (e.g. by explicit or implicit feedback) they must register information about the document and also about the time and the context in which it was interesting. Since a user profile content is created and updated based on these experiences we face the necessity of specifying what a user profile contains.

In order to get accurate and complete enough user models to fulfill the requirements of any textual-based agents, we take into consideration several aspects. First, user interests are usually related to multiple topics which also tend to be very specific. Thus, a natural way of organizing topics of interest is through a hierarchy of increasing specificity. This way of representation also facilitates the incremental learning process of the user profile. Second, typically user interests are not isolated, there are relationships among them that are established by the user himself. These relationships should be also represented in user profile content to allow an agent to take advantage of them (e.g. to recommend pages related to the one the user is currently reading, to arrange news about related topics close in a personal newspaper, etc.). Finally, user interesting topics are not equally relevant or interesting to a user at any time. A user profile needs to establish the different levels of relevance for each topic according to the temporal context in which the topic will be used. Based on this information, an agent will be able to avoid certain topics in given periods of time (e.g. news about sports in work hours) and also to recognize good opportunities to make suggestions about other topics (e.g. offer travel information before vacations). Figure 1 depicts the content of a user profile. It is composed of a hierarchy of topics, each of them with associated temporal contexts of relevance and a number of experiences. An example of profile content is shown in Figure 2.

The *experience manager* component is in charge of representing user experiences in a unified way in order to analyze them according to previous experiences and user interests. This component codified the representation method chosen for documents (e.g. TF-IDF vectors [12]). It requires to know what topic(s) the experiences belong to. Another component in the architecture, the *topic manager*, completes this function. It receives an experience and returns the topic(s) in the profile where this experience fits into with an associated degree of relevance. To accomplish this task this component gets topic hierarchical information from the *hierarchy manager* component, which maintains all the information related to the topic hierarchy.

The incorporation of a new experience from the user, is reported to three other components in the architecture besides being stored in the profile. Components notified of the arrival of a new experience are the *topic extractor*, the *relationship manager* and the *temporal context manager*.

The *topic extractor* component identifies topics of interests by abstracting them from several reading experiences with similar characteristics. Once a topic is inferred by this component it is sent to the *hierarchy manager* to be inserted into the hierarchy. New topics are also sent to the *topic manager* in order to update topic information.

The *relationship manager* is the component responsible for discovering and updating relationships among topics. It looks for relationships by observing a user browsing behavior (e.g. the user usually reads about *basketball* after reading about *football*), which is provided by the *observation* component. Starting from this information the *relationship manager* establishes when to add or remove relationships. In such cases, the *hierarchy manager* is notified to reflect the new knowledge.

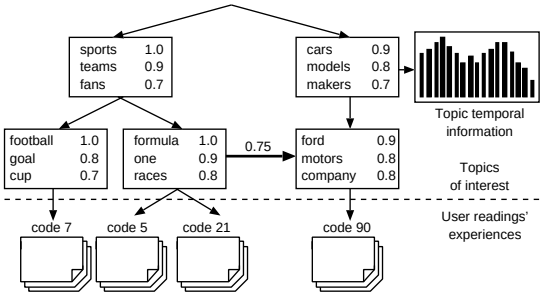

Figure 2: Example of a user profile content

The *temporal context manager* updates the information about routines of access to topics. Temporal data of a topic includes time and date of each of its readings. Thus, each experience received by this component is added as a new observation into the topic(s) this experience belongs to.

We have described so far, how experiences are used to create and update user profiles, the other components in the architecture are related to the actions an agent will be able to perform. Since our agents behave as assistants, they can either personalize tasks started by the users or take the initiative to offer them suggestions. For the first task, the *action* component asks about the relevance of a given piece of information (e.g. a Web page) in regard to a user interest and receive a numerical indicator representing the relevance level and also the topic(s) this information belongs to. For the second task, a component is required in the architecture to pro-actively identify opportunities to recommend information about certain topics and report about them to the *action* component. The *recommender* component in the architecture is in charge of this task, it asks for information from the *topic manager* and the *temporal context manager* components.

Finally, the *hierarchy comparator* component acts each time an agent is involved in a collaborative task with other agents. In this case, it is necessary to identify topics two users have in common in order to share information. This component receives either a complete hierarchy or only a set of topics belonging to another user and compares them with the actual profile content to return shared topics.

3 Architecture Components Overview

This section presents the main components of the user profiling architecture, their functionality and how they interact in order to learn and adapt user profiles.

3.1 Experience Manager Component

In order to allow agents to reason based on experience, the *experience manager* uses Textual Case-Based Reasoning (TCBR), an specialization of Case-Based Reasoning (CBR) for textual information. CBR is a problem solving paradigm that reuses solutions of previous experiences, which are named cases, in order to solve new problems. The term TCBR was subsequently coined for situations where these experiences are given by textual documents [5].

A case-based reasoner remembers problem-solving situations as cases, textual cases in TCBR, in a case base. Then it retrieves relevant cases (the ones matching the current problem) and it adapts their solutions to solve new situations. These solutions could be complex, like the description of a treatment for a given disease, or simple like the category in which a problem fits into. In the last perspective, CBR is applied to accomplish a classification task, i.e. find the correct class for an unclassified case. The class of the most similar past case becomes the

solution to the classification problem.

In the context of CBR a case contains the specific knowledge that describes a particular situation [4]. The main parts of a case are the description of the situation or problem that has been previously solved, the description of its solution and the outcome of applying the solution to the problem.

In our topic classification problem a case describes a particular user reading experience on the Web which the agent can detect as interesting for the user (e.g. based on explicit feedback). Each case holds the main features of a document, which enables the reasoner to determine its topic considering previous experiences. Cases solutions are the topic documents belong to. Thus, a user topic of interest is extensionally defined by the set of cases in the profile with the same solution. Using this approach, previous interesting documents can help to categorize new ones into specific categories of interests.

In order to completely describe a user reading experience, both the content of the document read by the user and the characteristic of the visit to that document need to be included in the case. Data about the visit includes the time spent reading the page in relation to its length, the page URL and the level of interest the user has in the document according to the agent criteria (i.e. based on a number of heuristic to determine the user interest on a given document).

The document content is described using a bag-of-words representation commonly used in the area of Information Retrieval (IR). To reflect the importance of each word in the document a weight is associated to each of them as the result of a function $weigth(w_j, d_i) = tf_{ij} + \Delta_{ij}$, where tf_{ij} is the frequency of a word w_j in the document d_i and Δ_{ij} an additive factor defined in terms of several word characteristics in the document. The value of Δ_{ij} is calculated taking into account the word location inside the document HTML structure (e.g. words in the title are more important than words in the document body) and the word style (e.g. bold, italic. etc.).

Previous to obtain a document representation, non-informative words such as prepositions, conjunctions, pronouns and very common verbs are removed by using a standard stop-word list. After stop-words removal a stemming algorithm is applied to the remaining words in order to reduce the variant forms of a word to a common one [10].

The solution of textual cases is the topic they belong to, among those topics existing in the profile. This topic need to be determined by the agent before adding the case to the profile. The *topic manager* component is consulted to get the solution of a case. It compares the case with other cases contained in the profile and the solution of the most similar case, if one exists, is applied to it. If there is not a similar case in the profile, it is assumed that the case belongs to a new topic representing this new interest.

Finally, a case outcome registers feedback to suggestions made starting from it. Since agent actions will be made based on previously recorded experiences, they hold the number or successes and failures when they were used. This information allows an agent to have a degree of confidence on each case to decide future actions or, eventually, determine when an experience is not longer valid.

3.2 Topic Extractor Component

In order to get the topic of a new experience or case, the *topic extractor* component needs to compare it with the other cases in the profile. Cases are compared through the number of dimensions used to describe the situations contained in them. Since each of these dimensions describes different aspects of the problem, a similarity function, *sim*, has to be defined to each of them. In our textual cases, dimensions to be compared are the URL and the list of relevant words describing the content of a document. The former depends on the host where the page is and the latter is calculated by the inner product with cosine normalization [12]:

$$sim(l_i, l_j) = \cos(\alpha) = \frac{\sum_{k=1}^{r} w_{ik} * w_{jk}}{\sqrt{\sum_{k=1}^{r} w_{ik}^2} * \sqrt{\sum_{k=1}^{r} w_{jk}^2}} \quad (1)$$

where l_i and l_j are the respective lists of words, and w_{ik} and w_{jk} the weights of word k within them. This similarity function measures the cosine of the angle α between the vectors representing both documents.

A numerical evaluation function that combines the

matching dimensions with the importance value assigned to each of them, is used to obtain the global similarity between a entry case, C_E, and a retrieved one, C_R (case in the profile). The function used in our technique is the following:

$$S(C_E, C_R) = \frac{\sum_{i=1}^{n} w_i * sim(f_i^E, f_i^R)}{\sum_{i=1}^{n} w_i} \qquad (2)$$

where w_i is the importance of the dimension i, sim_i a similarity function for this dimension, and f_i^E, f_i^C are the values for the feature f_i in the entry and retrieved case respectively.

If the similarity value obtained from S is higher than a given threshold δ the entry and retrieved cases are considered similar and then we can conclude that both are about the same topic. In this case, the solution of the retrieved case is assigned to the new situation. Otherwise, if there is not any case with a similarity value higher than δ, the reasoner can not give a solution to the case and a new topic of interest has to be created. If this happens, the *hierarchy manager* and the *topic manager* are notified about the new topic to update the profile.

Notice that the threshold δ determines the amount of topics or case groups to be held in the user profile. A low value of δ allows many cases to be grouped into a same topic, even when they were not so similar, while a high threshold will produce a large number of topics formed by only a few cases. Through experimentation we establish a value of $\delta = 0.6$, where a compromise between profile degree of detail and amount of topics is reached.

3.3 Topic Manager and Hierarchy Manager Components

A hierarchy of increasing specificity is used to organize topics within a profile. The analysis of user experiences to discover interesting topics is performed by the *topic extractor* component. The *hierarchy manager* is aware of the addition and the elimination of topics. It also maintains the hierarchical relationships among them.

The topic hierarchy could be seen like a tree. Each internal node in the tree holds features shared by their child nodes and the cases below it. For textual cases these features are words in the document content. Items without these features live in or below their sibling nodes. Leaf nodes hold cases by themselves.

The topic hierarchy needs to be built starting from scratch. To do this, as soon as new cases appear describing user interests they are grouped by similarity in the profile as we have already seen. Each of these groups represents a very specific topic of interest to a user. Then, a general inductive process automatically builds a classifier for this topic or category c_i by observing the features of cases that have been classified under c_i. Thus, the inductive process extracts the features a novel document should have in order to be classified under c_i.

A classifier for a category is a function $F_i : d_j \rightarrow [0,1]$ that, given a document d_j, returns a number between 0 and 1 that represents the evidence for the fact that d_j should be classified under c_i. This function also has a threshold τ such that $F_i(d_j) \geq \tau$ is interpreted as a decision to classify d_j under c_i, while $F_i(d_j) < \tau$ is interpreted as a decision of not to classify d_j under c_i.

Once a classifier is built by representing a topic in the hierarchy, new cases belonging to this topic (those with $F_i(d_j) \geq \tau$) will be placed below it creating new child groups. From these groups new classifiers will be obtained and added like child nodes of the first classifier, defining a hierarchy of them. Cases not belonging to any topic in a given level of the tree will be placed in this level inside an existent group of cases or will create a new group.

We used lineal classifiers which represent a category or topic as a vector $c_i =< w_{1i}, ..., w_{ri} >$where w_{ji} is the weight associated to the word j in the category i. The function F_i associated to each classifier is the cosine similarity measure (Equation 1).

Features composing a classifier are selected by using the *document frequency* measure over a group of cases. The document frequency $\#T_r(t_k)$ of a word w_k is the number of textual cases in a given topic in which this word occurs. This value is calculated for each unique word appearing in the cases defining the topic and those words whose $\#T_r(t_k)$ was higher than a given threshold, γ, will constitute the classifier for that user topic of interest. Since the addition of new

cases to a group changes the words characterizing it, we can only extract a classifier (with the set of words with $\#T_r(t_k) > \gamma$) when these words do not change after the incorporation of n new cases.

3.4 Relationship Manager Component

The *relationship manager* component discovers and establishes relationships among topics in a user profile. This component receives observed browsing sessions from the agent. We assume that a user's browsing pattern allows us to discover user established relationships among topics. In this way, if a user usually reads about a topic X after reading about Y, an agent can infer that a relationships between both topics may exist. Relationships are obtained by mining association rules among topics.

The formal description of the problem of mining association rules is the following [1]. Let $I = \{i_1, ..., i_m\}$ be a set of *items* and $T = \{t_1, ..., t_n\}$ a database of transactions, where each transaction t_i is a set of items such that $t_i \subseteq I$. An association rule is an implication $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = 0$. The association rule $X \Rightarrow Y$ has support s in T if s% of the transactions in T contain $X \cup Y$, i. e.

$$s(X) = \frac{|\{t_i | X \cup Y \subseteq t_i\}|}{|T|} \quad (3)$$

The rule $X \Rightarrow Y$ holds with confidence c if c% of the transactions containing X also contain Y, i. e.

$$c(X,Y) = \frac{|\{t_i | X \cup Y \subseteq t_i\}|}{|\{t_i | X \subseteq t_i\}|} \quad (4)$$

Given a minimum confidence threshold c_{min} and a minimum support threshold s_{min}, the problem of mining association rules is to find all association rules $X \Rightarrow Y$ in T that have support $s(X \cup Y) \geq s_{min}$ and confidence $c(X,Y) \geq c_{min}$.

The association rules discovery problem is defined over a collection of subsets from an item space. We considered topics of interest for a user as items in our problem of determining associations among user interests. Before any mining algorithm could be applied, sequences of these topics must be grouped into logical units representing transactions. All topics visited by a user during a single browsing session conform a user session. Since the topics are those present in the user profile at the moment the browsing took place, some visited pages may belong to unknown or uninteresting topics to the user. A transaction differs from a user session in the fact that the size of a transaction can range from a single topic to all of the topics in a user session, depending on the criteria used to identify transactions. Different techniques can be applied to partitionate a user browsing session in order to extract transactions. We used a time window that divides the user browsing session into time intervals no larger than a specified parameter. This method assumes that meaningful transactions have an overall average length associated with them.

This problem was extended to determine associations at the right level of a taxonomy. For this purpose, each transaction t_i is extended to include each ancestor of a particular item i_i or topic in the hierarchy. Then, the confidence and support for all possible association rules $X \Rightarrow Y$ where Y does not contain an ancestor of X are computed. Finally, all those association rules $X \Rightarrow Y$ that are subsumed by an ancestral rule $\widehat{X} \Rightarrow \widehat{Y}$ are pruned. Where itemsets $\widehat{X}, \widehat{Y}$ only contain ancestors or identical items of their corresponding itemsets in $X \Rightarrow Y$.

The following four steps summarize the discovery of relationships among topics:

1. Determine $T = \{i_1, .., i_n\}$ adding the ancestors of each item i_j
2. Determine support for all association rules $X \Rightarrow Y$, where $|X| = |Y| = 1$
3. Determine confidence for all association rules $X \Rightarrow Y$ that exceed s_{min} in step 2
4. Output association rules that exceed c_{min} in step 3 and that are not pruned by ancestral rules with higher or equal confidence and support

3.5 Temporal Context Manager and Recommender Components

In order to provide a personalized assistance, agents need to know not only the information a user wants to

read, but also the right moment to recommend about them. To take this new issue into consideration we add to the user profiling architecture a component to manage information about topics access routine, the *temporal context manager*. While this component maintain the information about access routines to topics, the *recommender* component is in charge of recognizing good opportunities to make suggestions to the user. This last component is consequently responsible for the pro-active behavior of an agent.

Information about routines of access to topics is held in the form of time series, i.e. sets of quantitative data obtained in regular periods of time. From the time series analysis the *recommender* is able to: (a) identify the nature of the phenomenon represented by the sequence of observations, and (b) forecast (predict future values of a time serie variable).

Since experiences about a given topic are recorded as cases, agents can summarize this information in terms of the amount of access within hours, days or months. Starting from this data they will be able to detect behavioral patterns such as: readings on *sports* are mostly made at night while *finances* articles are read on work hours, first days of the month are preferred to visit shopping pages and every January the user shows interest in travel information.

In order to calculate the probability of a topic reading in a given period of time (hour/day/month), the *recommender* gets the average of readings in the same period according to previous data. For example, it can estimate the amount of readings about a topic *X* between 9 and 10 A.M. in order to establish how many pages about this topic should be recommend to the user. Moreover, taking into account this probability an agent can fairly distribute the number of suggestions to make in a given moment, e.g. the percentage of pages to suggest about each topic considering the whole set of topics in a user profile.

4 An Architecture Materialization

The user profiling architecture was applied to the development of the *PersonalSearcher* [3], an agent that assists users to find interesting documents on the Web. This agent searches the most most popular Web search engines in parallel and filtering the resultant documents according to the user topics of interest.

PersonalSearcher learns a model of topics of interest for his associated user based on the observation of user browsing on the Web. For each reading in the standard browser the agent observes a set of indicators in order to estimate the user interest in that Web page. This process is called implicit feedback since it can be obtained from the user without disturbing his normal behavior or distracting him to ask explicit evaluations for each visited page. These indicators are the time consumed in reading (with respect to its length), the amount of scrolling in a page and whether it was added to the list of bookmarks. Documents classified as interesting are captured as experiences by the *observation* component and sent to the corresponding component in the architecture.

Users interact with their *PersonalSearcher* expressing their information needs by keywords as usual. The agent posts these queries to the most popular search engines on the Web (Altavista, Infoseek, Excite, etc.) getting a set of documents that covers a wide portion of the Web. The relevance degree of each document in relation to the user profile is computed by *PersonalSearcher* in order to determine the convenience of suggesting the document to the user for a future reading. This process involves contacting the *topic manager* component in the user profiling architecture to ask about each document relevance. The result is the topic(s) documents belong to, and documents relevance in respect of each of them. Only documents that surpass a given threshold of similarity in regard to some topic in the user profile are sent back to the user as a result of his query. This relevance threshold could be adjusted from the user interface of the *PersonalSearcher* within a interval between 0 (all documents will be recommended) to 1 (any document will be recommended).

Figure 3 shows an example of the kind of assistance that could be expected from *PersonalSearcher*. The first window shows the result of a Web search using the keyword *agents*. The resultant list of documents includes pages about *software agents*, *travel agents*, *insure agents*, etc. If the user performing the search is interested in *software agents*, a number of reading experiences about this topic in their user profile are expected. Based on this profile the suggestions that

could be seen in the second windows in the figure, are mostly related to *software agents.* Pages about any other topic were considered by the agent as uninteresting to the user.

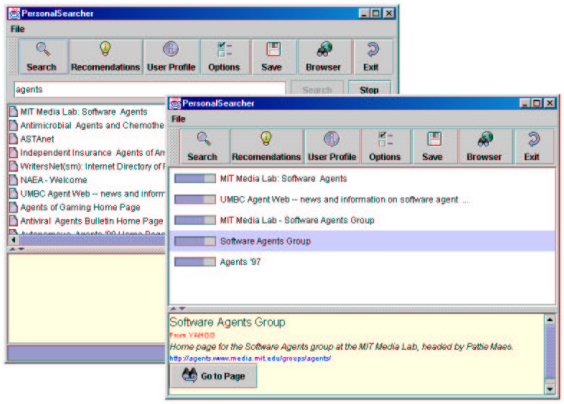

Figure 3: PersonalSearcher Agent GUI

We evaluate the *PersonalSearcher* effectiveness to recommend Web pages during this search based on the standard precision and recall measures from information retrieval [12]. From the user point of view, precision measures the proportion of relevant documents in those recommended by the agent while recall is the proportion of relevant documents in the search results that were in fact recommended to the user. As it can be observed, we require to know the relevance for each document in a search result in order to calculate precision and recall values. For this reason we used in this experiment a list of 200 Web pages marked as relevant or irrelevant by the user according to his own judgment.

Figure 4 shows the average of precision and recall reached by the agent along different values of the relevance threshold. The figure also shows variations in precision/recall value for different user profile sizes (amount of experiences/cases). This includes the effectiveness achieved by the agent after 20, 40 and 60 user experiences. It is worth noting the improvement of the agent effectiveness as the number of experiences in the user profile grows.

5 Related Works

A number of personal assistants that helps users in textual-based tasks have been built in the last decade. Some recent developments include Letizia [6], Syskill&Webert [9] and WebMate [2]. Most of these assistants use different models to represent documents coming either from the information retrieval or the machine learning area. Letizia and Syskill&Webert assist users browsing the Web using TF-IDF vectors [12] and a naive Bayes classifier respectively. WebMate generates personal newspapers based on multiple TF-IDF vectors each of them representing a topic of interest. The development of these agents was focused on the accomplishment of some specific tasks, such as browsing assistance and news filtering, but these techniques can not be reused in further agent developments with additional requirements. Instead, the user profiling architecture we have presented supports the development of capabilities shared by the majority of textual-based agents, such as determining the relevance of a given piece of information, pro-actively suggesting new information and establishing common information interests among a group of users.

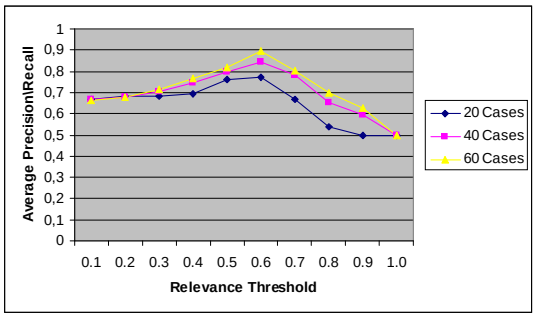

Figure 4: Experimental results

It is important to highlight the different scope of the user profiling architecture with respect to other generic agent architectures such as BDI [11] or InterRap [8]. In this sense, the last ones prescribe agent components and interactions at a higher level of abstraction than our architecture. They establish, for example, how interactions among a user profiling component and other components (the ones in charge

of perception, action and communication) take place. However, they do not prescribe how an agent learns, adapts and uses profiles in textual applications.

6 Conclusions

We presented in this paper a user profiling architecture to support the development of textual-based agents. To specify this architecture we determined the minimal content of profiles in order to fulfill most of the requirements of information agents. Based on this specification we designed the components in the architecture to acquire, adapt and use the knowledge contained in profiles.

The advantages of our architecture are twofold. First, it reduces the burden of developing agents involved with textual-based tasks by providing a generic design of software components and the interactions among them. As a consequence of the usage of this design, developers not only reduce time and effort, but also borrow our experiences on agent development. Second, the architecture describes the inner workings of the components in an application-dependent way, thus enabling the developer to tailor them to his specific requirements.

Finally, we implemented a search agent based on the architecture in order to validate it. Experimental results have shown how the effectiveness of this agent to filter search results improves as its knowledge about the user increases. Further experiments need to be conduced to evaluate the architecture behavior according to different agent requirements. Agents in other domains will be used to such a purpose as well as agents involved in collaborative tasks.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databasess. In *Proceedings of ACM SIGMOD Int. Conference on Management of Data*, pages 207–216, 1993.

[2] L. Chen and K. Sycara. WebMate : A Personal Agent for Browsing and Searching. In *Proceedings of the* 2^{nd} *Int. Conference on Autonomous Agents*, pages 132–139. ACM Press, 1998.

[3] D. Godoy and A. Amandi. PersonalSearcher: An Intelligent Agent for Searching Web Pages. In *Advances in Artificial Intelligence*, volume 1952 of *Lecture Notes in Computer Science*, pages 43–52. Springer, 2000.

[4] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[5] M. Lenz. Textual CBR and Information Retrieval – A Comparison. In *Proceedings 6th German Workshop on CBR*, 1998.

[6] H. Lieberman. Letizia: An Agent that Assists Web Browsing. In *Proceedings of the* 14^{th} *Int. Joint Conference on Artificial Intelligence*, pages 924–929. Morgan Kaufmann, 1995.

[7] P. Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31–40, 1994.

[8] J. Müller. *The Design of Intelligent Agents: A Layered Approach*, volume 1177 of *Lecture Notes in Artificial Intelligence*. Springer, 1996.

[9] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill&Webert: Identifying Interesting Web sites. In *Proceedings of the* 13^{th} *National Conference on Artificial Intelligence*, volume 1, pages 54–61. AAAI Press, 1996.

[10] M. Porter. An Algorithm for Suffix Stripping Program. *Program*, 14(3):130–137, 1980.

[11] A. S. Rao and M. P. Georgeff. Modeling Rational Agents within a BDI-Architecture. In *Proceedings of the* 2^{nd} *Int. Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484, 1991.

[12] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[13] S. Soltysiak and B. Crabtree. Knowing Me, Knowing You: Practical Issues in the Personalisation of Agent Technology. In *Proceedings of the* 3^{rd} *Int. Conference on Practical Application of Agents and Multi-Agent Technology (PAAM'98)*, 1998.