A User Profiling Architecture for Textual-Based Agents

Daniela Godoy[1]and Analía Amandi

ISISTAN, Facultad de Ciencias Exactas,
Universidad Nacional del Centro de la Prov. de Buenos Aires,
Campus Universitario, Paraje Arroyo Seco (BBO7001B), Tandil, Bs. As., Argentina
[1]Also CONICET
{dgodoy, amandi}@exa.unicen.edu.ar

Abstract Several intelligent agents have been developed in the last decade to help users with the increasing volume of information available in the WWW. Despite the efficiency of these agents depends 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 specify each time, not only the user profiles content, but also how to acquire it and how to adapt it to changes on user interests. In this work we present an architecture which prescribe 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 report in this work.

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's interests, habits and preferences. Agents developed in this area have addressed many tasks, such as searching and browsing the Web [8,9,11] and electronic news filtering [15].

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 users' 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 use to assist users [16].

For the majority of the agents performing textual-based tasks currently in existence, the information contained within profiles is simply the set of interests a user has and, in some cases, also his dislikes. These interests regarding topics or subjects of interest are represented for each agent according to its own approach at different levels of detail. Consequently, profiles 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 profiles 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 been 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 it was captured (e.g. time and date, user tasks context, etc.), thus loosing contextual information about user interests. In contrast, those approaches from the second group can easily capture contextual information but its application to textual data is not straightforward [17]. 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 to each group of approaches need to be solved by agents 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 users, but also how to obtain this knowledge and how to use it later to provide a personalized assistance. Moreover, they also need both to identify which parts of a user profile content will be affected as the user behavior change 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 of 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 in the Web. It could be applied in the construction of, for example, browsing assistants, search agents, knowledge discovery 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 users community.

In order to evaluate the architecture we implemented a search agent, named *PersonalSearcher*, based on it. This agent assists users to find interesting documents in 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—together with a description of the interactions between these

components—the connectors [4]. 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, its is show 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.

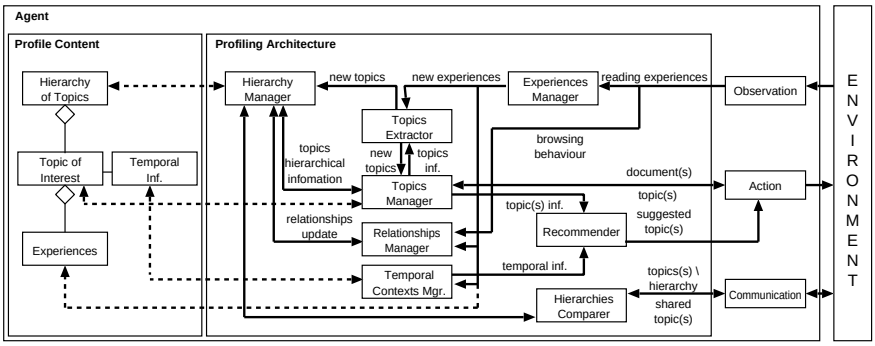


Figure 1. User Profiling Architecture

Agents observe events on the environment (e.g. user activities), understands what it perceives by reasoning about it, and executes actions depending on its current mental state. Since *observation* and *action*, as much as *communication*, depends on the application being developed, these tasks 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 [13] or InterRap [10].

The agent *observation* component is required to collect experiences about interesting documents read by the user. Regardless 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 resulted interesting. Since a user profile content is created and updated based on these experiences we faced the necessity of specifying what a user profile will contain.

For this task we take into account several aspects to get accurate user models and also complete enough to fulfill the requirements of any textual-based agent. First, users interests are usually related to multiple topics which also tend to be very specific. Thus, a natural way of organize interesting topics 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 will be also represented in user profiles 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 closely in a personal newspaper, etc.). Finally, users' interesting topics are not equally relevant or interesting to a user on 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, at the same time, recognize good opportunities to make suggestions about other topics (e.g. offer travel information previous to the 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 a user profile content is show in Figure 2.

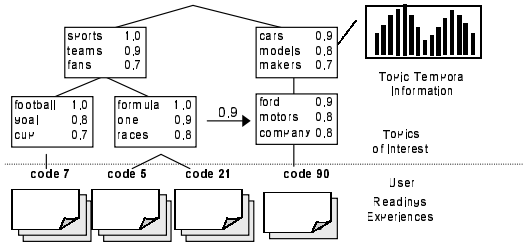


Figure 2. Example of a user profile content

The *experiences manager* component is in charge of representing users' experiences in a unified way in order to analyze them according to previous experiences and users' interests. This component codify the representation method chosen for documents (e.g. boolean or TF-IDF vectors [14]). It requires to know what topic(s) the experiences belong to. Another component in the architecture, the *topics manager*, completes this function. It receives a experience and returns the topic or topics in the profile where this experience fits into with a degree of relevance respect to each of them. To accomplish this task this component gets topics hierarchical information from the *hierarchy manager* component, which maintain all the information related to the topics hierarchy.

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

The *topics 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 send to the *hierarchy manager* to be inserted into the hierarchy. New topics are also send to the *topics manager* in order to update topics information.

The *relationships manager* is the component responsible for discovering and updating relationships among topics. It looks for relationships by observing a user's browsing behavior (e.g. the user usually reads about *basketball* after reading about *football*), which is provided by the *observation* component of the agent. Starting from this in-

formation the *relationship manager* establishes when to add or forget relationships. In such cases, the *hierarchy manager* is notified to reflect the new knowledge.

The *temporal contexts manager* updates the information about routines of access to topics in the profile. Temporal data related to a topic include time and date of each of its readings. Thus, each experience received by this component is added as a new observation corresponding to the topic(s) this experience belong to.

Until this moment we described how experiences are use to create and update users' profiles, the others 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 for the users or take the initiative to offer them suggestions. For the first task, the *action* component asks about the relevance of a given information (e.g. a Web page) in regards to a user's interests 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 inform about them to the *action* component. The *recommender* component in the architecture is in charge of this task. To do so, it asks for information from the *topics manager* and the *temporal contexts manager* components.

Finally, the *hierarchies comparer* component acts each time an agent is involved in a collaborative task with other agents. In this case, it is necessary to identify topics that 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 compare them with the actual profile content to return topics common for both users.

3 Architecture Components Overview

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

3.1 Experiences Manager Component

In order to allows agents to reason based on experience, the *experiences 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 [7].

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 concept or 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. Thus, a category is extensionally defined by the set of cases sharing the same solution.

In the context of CBR a case contains the specific knowledge that describes a particular situation [6]. 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 or results of applying the solution to the problem.

In our topic classification problem a case describes a particular user's reading experience in the Web which the agent can detect as interesting (e.g. based on explicit feedback). Each case hold the main features of a document, which enable the reasoner to determine its topic considering previous experiences. Cases solutions are the topics documents belong to. Thus, a user's 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 a level of interest the user has in the document according to the agent criteria (i.e. based on a number of heuristic an agent can 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 with 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 [12].

The solution of textual cases is the topic which they belong among those topics existent in the profile. This topic need to be determined by the agent before adding the case to the profile. The *topics manager* component is consulted in order to get the solution (topic) 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 case similar in the profile, it is assumed that the case belong to a new topic representing this new interest.

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

3.2 Topics 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 on them. Since each of these dimensions describe different aspects of the problem, a similarity function, *sim*, need 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 [14]:

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

where l_i and l_j are the respective lists of words, and w_{ik} and w_{jk} the weights of the word k in each list. 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:

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

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 *topics manager* are notified about the new topic to update the profile.

Notice that the threshold δ determinate the amount of topics or cases groups to be held in the user profile. A low value of δ allows many cases to be grouped together into a same topic, even when they were not so much 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 among 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 of interest in a user profile. The analysis over user experiences (cases) to discover topics of interest is performed by the *topic extractor* component as we previously explained. The *hierarchy manager* is aware of the addition and 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 conformed by words in the document content. Items without those features live in or below their sibling nodes. Leaf nodes hold cases 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 previously described. Each of these groups represents a very specific topic of interest of a user. Then, a general inductive process automatically builds a classifier for this subject or category c_i by observing the features of a set of cases that have been classified under c_i. Starting from these features, the inductive process extract 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 classifying d_j under c_i, while $F_i(d_j) < \tau$ is interpreted as a decision of not classifying 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 (those with $F_i(d_j) < \tau$) 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's 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 Relationships Manager Component

The *relationships manager* component discovers and establishes relationships among topics in a user profile. This component receives from the agent observed browsing sessions. 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 among 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 literals called *items*. Let $T = \{t_1, ..., t_n\}$ 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.

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

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

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

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 interesting for a user as items in our problem of determining associations among users' 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's profile at the moment the browsing took place, some visited pages may belong to unknowns or uninteresting topics to the user. A transaction differs from a user session in 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 partitioning a user browsing session in order to extract transactions. We used a time window that divide 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 extend to also 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 Contexts Manager and Recommender Components

In order to provide a personalized assistance, agents need to know not only the contents a user wants to read, but also the right moment to recommend about it. 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 contexts* 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 hold 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 of: (a) identifying the nature of the phenomenon represented by the sequence of observations, and (b) forecasting (predicting future values of a time series variable).

Since experiences about a given topic are recorded as cases, agents can summarize the information contained on them in terms of the amount of access within hours, days or months. Starting from this information 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, firsts days of the month are preferred to access to shopping pages and every January the user shows interest on travel information.

In order to calculate the probability that a user read about a topic 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 recommend to the user. Moreover, taking into account this probability an agent can fairly distribute the number of suggestions to made in a given moment, e.g. the percentage of pages to suggest on each topic considering the whole set of topics in a user profile.

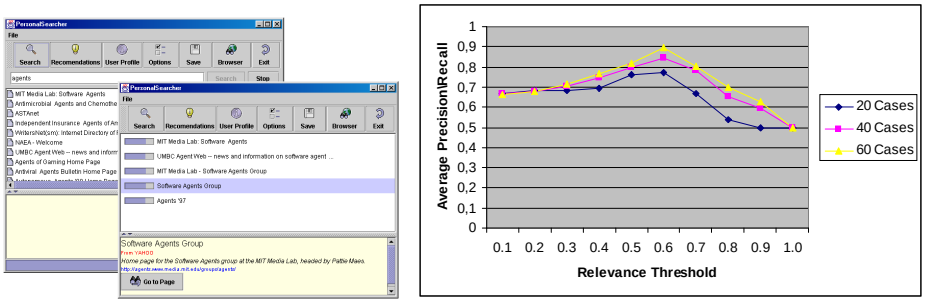
4 An Architecture Materialization: The *PersonalSearcher* Agent

The user profiling architecture was applied to the development of an agent named *PersonalSearcher* [5] that assists users to find interesting documents in the Web. This agent carries out a parallel search in the most popular Web search engines and filters the resultant list of documents according to the user topics interest.

PersonalSearcher learns a model of topics of interest for his associated user based on observation of user browsing in the WWW. 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 relation to its length), the amount of scrolling in a page and whether it was added to the list of bookmarks. Documents classified as interesting by this means are captured as experiences by the *observation* component and then send to the corresponding component in the user profiling 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 in 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* to determine the convenience of suggesting the document to the user for a future reading. This process involves contacting the *topics manager* component in the user profiling architecture to ask about each document relevance. The answer is the topic(s) documents belonging to and its relevance respect to them. Only documents that surpass a given threshold of similarity as regards some topic in the user profile are sent back to the user as a result to 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).



(a) PersonalSearcher Agent GUI (b) Experimenal results

Figure 3(a) 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. Since the user performing the search is interested in *software agents* he possesses a number of reading experiences about this topic in their user profile. 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 considering by the agent as uninteresting to the user.

We evaluate the effectiveness of the *PersonalSearcher* agent to recommend Web pages during this search based on the standards precision and recall measures from information retrieval [14]. 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 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 3(b) shows the average of precision and recall reached by the agent along different values of the relevance threshold. The figure also shows variations in pre-

cision/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 help users in textual-based tasks have been built in the last decade. Some recent developments include Letizia [8], Syskill&Webert [11], WebMate [2] and NewT [15]. 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 [14] and a naive Bayes classifier respectively. WebMate generate personal newspapers based on multiple TF-IDF vectors each of them representing a topic of interest. Finally, NewT uses genetic algorithms to filter USENET news. The development of these agents was focused in 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 determine the relevance of a given piece of information, pro-actively suggest new information and establish common information interests among a group of users.

It is worth noting the different scope of the user profiling architecture with respect of generic agent architectures such as BDI [13] or InterRap [10]. In this sense, the last ones prescribe agents' 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 learn, adapt and use profiles in textual applications.

6 Conclusions

We presented in this paper a user profiling architecture to support the development of agents for textual-based tasks. To specify this architecture we determined the minimal content of profiles in order to fulfill most of the requirements of common information agents. Based on this specification we designed the different 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, for example for a Web searcher agent or a Web pages recommender.

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 regarding different agents requirements. Another agent based on this architecture, the *NewsAgent* agent [3], will be used to such a purpose. This agents generates personal newspaper for a user based on his preferences. Since news are very dynamic, we consider this agent more suitable to evaluate how adaptation performs in our architecture.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 207–216, 26–28 1993.
2. L. Chen and K. Sycara. Webmate : A personal agent for browsing and searching. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 132–139. ACM Press, May 1998.
3. D. Cordero, P. Roldán, S. Schiaffino, and A. Amandi. Intelligent agents generating personal newspapers. In *Proceedings del ICEIS99, International Conference on Enterprise Information Systems*, Setubal, Portugal, March 1999.
4. D. Garlan and M. Shaw. An introduction to software architecture. In *Advances in Software Engineering and Knowledge Engineering*, pages 1–39, 1993.
5. D. Godoy and A. Amandi. PersonalSearcher: an intelligent agent for searching web pages. In *Proceeding of the international joint conference IBERAMIA'2000*, volume 1952 of *Lecture Notes in Artificial Intelligence*, pages 43–52, Brazil, November 2000. Springer-Verlag.
6. J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, CA, 94403, 1993.
7. M. Lenz. Textual CBR and information retrieval – a comparison –. In *Proceedings 6th German Workshop on CBR*, 1998.
8. H. Lieberman. Letizia: an agent that assists web browsing. In *Proceedings of the fourteenth international joint conference on artificial intelligence (IJCAI'95)*, pages 924–929. Morgan Kaufmann, August 1995.
9. D. Mladenic. Personal WebWatcher: Implementation and design. Technical Report IJS-DP-7472, Department for Intelligent Systems, J.Stefan Institute, 1996.
10. J. Müller. *The design of intelligent agents: a layered approach*, volume 1177 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag Inc., New York, NY, USA, 1996.
11. 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, August 1996.
12. M. Porter. An algorithm for suffix stripping program. *Program*, 14(3):130–137, 1980.
13. A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484, April 1991.
14. G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.
15. B. D. Sheth. A learning approach to personalized information filtering. Master's thesis, Massachusetts Institute of Technology, January 1994.
16. S. Soltysiak and B. Crabtree. Knowing me, knowing you: Practical issues in the personalisation of agent technology. In *Proceedings of the (PAAM'98)*, 1998.
17. G. Webb, M. Pazzani, and D. Billsus. Machine Learning for User Modeling. *User Modeling and User-Adapted Interaction*, 11:19–29, 2001.