Journal of Intelligent Information Systems, 25:3, 251–273, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

An Interface Agent Approach to Personalize Users' Interaction with Databases

SILVIA SCHIAFFINO* sschia@exa.unicen.edu.ar
ANALÍA AMANDI* amandi@exa.unicen.edu.ar
ISISTAN Research Institute, Fac. Ciencias Exactas, Univ. Nac. del Centro de la Pcia. Bs. As., Campus Universitario, Paraje Arroyo Seco, Tandil, 7000, Bs. As., Argentina

Received June 26, 2003; Revised February 17, 2004; Accepted September 22, 2004

Abstract. Making queries to a database system through a computer application can become a repetitive and time-consuming task for those users who generally make similar queries to get the information they need to work with. We believe that interface agents could help these users by personalizing the query-making and information retrieval tasks. Interface agents are characterized by their ability to learn users' interests in a given domain and to help them by making suggestions or by executing tasks on their behalf. Having this purpose in mind we have developed an agent, named *QueryGuesser*, to assist users of computer applications in which retrieving information from a database is a key task. This agent observes a user's behavior while he is working with the database and builds the user's profile. Then, *QueryGuesser* uses this profile to suggest the execution of queries according to the user's habits and interests, and to provide the user information relevant to him by making time-demanding queries in advance or by monitoring the events and operations occurring in the database system. In this way, the interaction between database users and databases becomes personalized while it is enhanced.

Keywords: interface agents, personalization, user profiling, Bayesian networks, databases

1. Introduction

The legacy of the revolution in technologies and communications that took place in the last decade is a great volume of information available in a wide variety of electronic media. A big part of this information is stored in relational databases. The incredible growth of the WWW made companies leverage their database systems to Internet and intranet contexts. Consequently, managing the information contained in databases has become even a more complex and time-demanding process for users of such systems (Han and Sterling, 1997).

For example, consider a group of technicians working with a LIMS (Laboratory Information Management System) in a petrochemical plant. These technicians are in charge of tracking and analyzing information about samples of different products, which are originated in several sectors within the plant. This information is stored in the laboratory database system and it can be accessed through the LIMS. Daily, these technicians make queries to the LIMS to get the information they need. Making queries through the LIMS is a routine task for these technicians, since they make similar queries in different opportunities to get the

*Also CONICET, Comisión Nacional de Investigaciones Científicas y Técnicas, Argentina.

information they usually work with. Each user makes queries depending on his information needs, his interests and the tasks he carries out in the plant. For example, an analyst may be interested in detecting the causes of rejection of samples, while other analyst may be interested in the whole sample analysis process of a particular product. We can observe that the task of making queries is different for different users, but repetitive for each of them. Having some kind of facilities in database systems that provide personalized services would be quite convenient in this situation.

Personalization is tool-box of technologies and application features used in the design of an end-user[1] experience. The main goal of personalization is providing value to the end user. Personalization involves a process of gathering user-information during interaction with the user, which is then used to provide appropriate assistance or services, tailor-made to the user's needs (Bonett, 2001). In our domain, and following the example described in the previous paragraph, the analysts' work would be alleviated if the system could inform them when samples are rejected or when samples change their states within the system. Moreover, the system would save users' time and effort if it could discover their information needs and working habits and offer them the information they need before they have to ask the system for it by, for example, making the necessary queries in advance.

Unfortunately, current database management systems do not provide this kind of personalization. Some personalization, or properly said customization, can be achieved with database applications, such as filtering the access of users to a set of tables the user is supposed to work with. A distinction between customization and personalization should be made. Customization occurs when the user can configure an application manually, adding and removing elements in his configuration according to his needs and preferences. In personalization, the user is seen as being more passive, while the system monitors, analyzes and reacts to users' behavior, although users can still configure some parameters. Using customization, as many information systems and database applications do, it is not possible to discover the different interests or needs each user has regarding the data stored in the database, how these interests change over time and the habits or routines users exhibit when working with the database application. In this work, we aim at personalizing users' interaction with database systems, as it has been done in other areas where a user's interests or needs are represented by queries (Wen et al., 2002; Lenz et al., 1998).

A technology widely used to provide this type of personalization is interface agent technology. Interface agents are computer programs that have the ability to learn a user's preferences and working habits, and help him to deal with one or more computer applications, reducing in this way the user's workload (Maes, 1994). Interface agents have been applied in a wide variety of domains, such as information filtering and information retrieval (Billsus and Pazzani, 1999), e-commerce (Morris et al., 2000), web browsing assistance (Lieberman et al., 2001), e-mail management (Segal and Kephart, 2000) and meeting scheduling (Mitchell et al., 1994).

Interface agents can be very helpful to database users for whom making queries to a database system through a computer application is a repetitive and time-consuming task. In this work, we propose a solution to personalize users' work with databases. We have developed an interface agent named *QueryGuesser* that acts as a user's personal assistant. This agent observes a user's behavior while he is working with a database application and

learns his information needs or interests, and also his querying habits or working routines. This information constitutes the user's profile. *QueryGuesser* then uses this profile to suggest the execution of queries according to the user's habits and interests, and to provide the user information relevant to him by making time-demanding queries in advance or by monitoring the events and operations occurring in the database system. In this way, the interaction between users and databases is both personalized and enhanced, since each user will have access to the information he needs when he needs it.

However, deciding how to build a database user profile was not an easy task for us. This paper describes our approach to build a database user profile and explains how an interface agent uses it to provide assistance to a database user. In our approach, *QueryGuesser* has to achieve two goals: discovering a user's topics of interests from the user's queries, and discovering a user's querying habits by analyzing his querying behavior.

The rest of this paper is organized as follows. Section 2 presents an overview of the *QueryGuesser* agent. In Section 3 we discuss how a set of *QueryGuesser* agents can assist a community of users. Section 4 describes the contents of a user profile in our domain. Section 5 describes how *QueryGuesser* builds a user profile. Section 6 presents our experimental results. Section 7 describes some related works. Finally, Section 8 presents our conclusions and future work.

2. The *QueryGuesser* agent: An overview

Interface agents are computer programs that learn a user's preferences and working habits and help him to deal with one or more computer applications, reducing in this way the user's workload. A commonly used metaphor for understanding interface agent paradigm is comparing them to a human secretary or personal assistant who is collaborating with the user in the same work environment (Maes, 1994). Initially, a personal assistant is not very familiar with the habits and preferences of her[2] employer and may not be very helpful. The assistant needs some time to become familiar with the particular working habits of her boss. However, the assistant becomes gradually more effective and competent as she acquires knowledge about him. This knowledge can be acquired by observing how the employer performs tasks, by receiving explicit instructions and feedback from the employer, by asking him for information or by learning from other assistants' experience. Then, the assistant can perform tasks that were initially performed by the employer, she can suggest him the execution of tasks, she can notify the employer about interesting situations and warn him about problems that may arise. In the same way, an interface agent can become more competent as it interacts with a user and learns about him.

QueryGuesser is an interface agent that assists users who work with a relational database system through Web or intranet applications. Its functionality is depicted in figure 1. The *QueryGuesser* agent has the capability of personalizing the interaction between a user and an application through which the user makes queries to a database system. *QueryGuesser* observes a user's behavior while he is making queries to a database system through the application interface. The agent records the queries and uses this information to build the user profile. As shown in the figure, the profile comprises both the different topics the user is interested in and his querying routines. *QueryGuesser* then uses this profile to provide

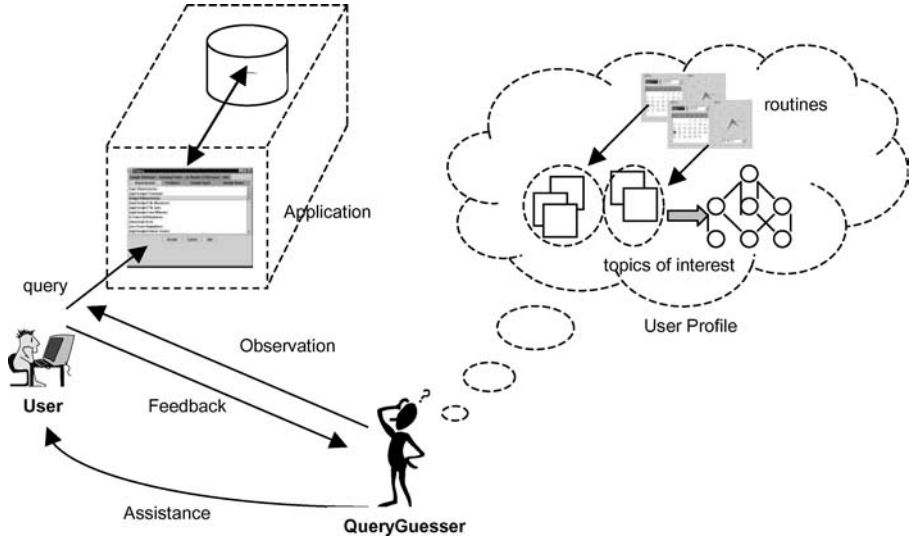

Figure 1. *QueryGuesser* overview.

assistance to the user with his computer work. In turn, the user can evaluate the agent's actions by providing explicit feedback. Besides, the agent observes the user's actions after assisting him to capture some implicit feedback. The user feedback is used to update the user profile so that the agent can provide better assistance to the user in the future.

For example, consider a user of LIMS named John Smith who frequently makes queries to the lab database looking for information. This database stores information about products, sampling points, states of samples, different types of samples, analysis of samples and product specifications. John Smith is in charge of controlling the state of the samples of various products, mainly of those taken at the *Storage*, the *Polymerization* and the *Arnipol* departments. To obtain this information, he makes queries hourly to discover which samples have been rejected, and daily to find out which have finished the analysis process. He is particularly interested in propane and arnipol samples. *QueryGuesser* can learn this user's interests by observing the queries he makes and when he makes them. The assistant agent can use this information to assist the user in different ways. First, *QueryGuesser* can notify John about information relevant to him while he is working with the LIMS. For example, suppose that the user has made a query at 5 pm and there are no rejected samples in the lab. Fifteen minutes after this query, information about two rejected samples becomes available. Thus, the agent notifies the user about these samples, keeping the user well-informed and up-to-date. To achieve this goal, the agent monitors the application in order to detect when information related to the user's information needs and interests is added or removed. Second, the agent can present the user a report containing interesting information regarding the user profile. For example, *QueryGuesser* can prepare a report about those samples having a "finished" state and show it to the user when he enters into the system. To achieve this goal, the agent has to make one or more queries on the user's behalf, saving in

this way the user's time. Third, the agent can make queries on the user's behalf according to his routine. To do this, the agent has to learn when the user makes a certain query, i.e. find patterns in his querying behavior. In our example, the agent can learn that John makes the same query every hour to find out about rejected samples, and it can automate this process. In turn, John Smith can also make new queries, without interacting with *QueryGuesser*.

To build a user profile in this domain *QueryGuesser* has to carry out two challenging tasks: discovering the interests of a user regarding the data stored in the database and discovering his querying habits. The only sources the agent has to obtain this information are the queries the user makes through a given application, a LIMS in our example. Thus, *QueryGuesser* observes a user's behavior while he is making queries to a database system through an information system and records some specific data about each query the user makes: the features or keywords used by the user to make the query and temporal information describing the moment in which the query was executed (e.g. date, time, shift). The features or keywords appearing in a user's queries provide evidence of the user's interests regarding the database, since these keywords reveal his information needs. The date and time enable the agent to discover behavioral patterns.

Each topic of interest is described by a set of features and the values these features take, as well as an importance value or weight associated with each feature and feature value. Besides, each topic comprises the relationships that exist among its components, represented by the network in figure 1.

The user profile also comprises a user's working habits or routines, which are described by a set of situations in which the user makes queries. Each routine is associated with one or more topics of interest, and vice versa. Section 4 describes in detail the two components of a user profile in our domain.

QueryGuesser uses the user profile to assist the user with his work. *QueryGuesser* basically helps the user in the following ways: it can suggest the user the execution of queries according to his interests and needs, it can perform some of these queries in advance in order to save the user's time, and it can alert the user about new relevant information. Then, the agent processes the feedback the user provides regarding the assistance it has given him. The user can explicitly evaluate the quality and relevance of the queries the agent has suggested him and the behavioral patterns the agent has inferred. Besides, the agent can observe the user's behavior after assisting him to detect some indirect (or implicit) feedback, such as the user making a query different from those the agent has suggested. The user feedback is used to improve and update the user profile.

3. A set of *QueryGuesser* agents assisting a set of users

In several domains such as a LIMS, a common scenario is having a set of users working with a computer application through which they connect to a (distributed) database to get the information they have to work with or they are interested in. Figure 2 depicts this situation. Each user makes queries to the database through the LIMS interface and the results are presented to him through another interface. When interface agent technology is used, each user has a personal assistant helping him, as shown in the figure. Each *QueryGuesser* agent

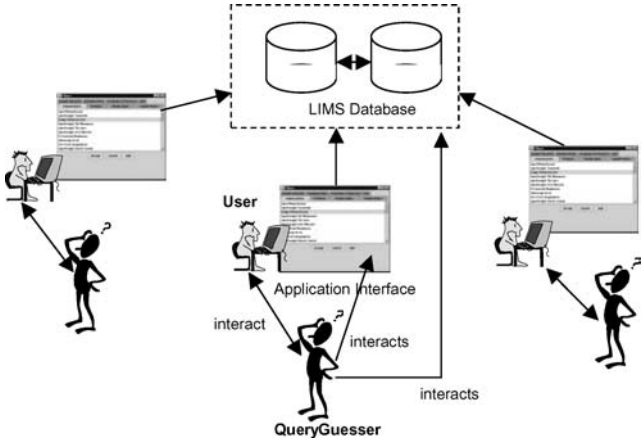

Figure 2. Assisting a set of users.

assists its user according to the profile it builds from the observation of the user's behavior and from the interaction between the user and the agent.

A potential source of learning for interface agents is asking other agents for advice when they do not know how to assist their users. In such a situation, the agents need some mechanism to compare the profiles of the users they are assisting, and if they are similar, the assistance provided to a user can be useful to assist a second user. Nevertheless, in our domain it is not common that different users show a similar behavior and have the same interests, unless they do the same job. This can be the case of users doing shift work and having the same role within a company. Only in this situation a user's profile could be used to assist another user, because the agents know that they share the same information needs.

However, a key aspect when working with multiple agents assisting multiple users is information privacy. It is the user decision if he wants his profile to be public or not. For example, the manager of a company surely does not want his employees to know what kind of queries he usually makes and what information he usually gets from the LIMS database. Due to privacy concerns, the agents in our approach do not share information with each other and each user profile is private.

4. A database user profile

The knowledge the agent gathers about a user is processed to build the user profile. A profile is a description of someone containing all the most important or interesting facts about him. In our context, a user profile contains all the information an agent needs to provide personalized assistance to a user of a database application.

A user profile often contains both application dependent and application independent information about a user, and its content varies from one application domain to another. Application independent information includes mainly personal information about the user,

such as name, hobbies, job, the department where he works and the section where he works. Application dependent information includes a user's interests, preferences, goals, working habits, behavioral patterns, knowledge, needs, priorities and commitments regarding a particular domain or application.

We define the application dependent part of a database user profile as the set of topics the user is interested in, which motivate him to make queries to the database system through the LIMS, together with the user's querying habits or routines, i.e. when queries are made.

$$Profile = Topics \cup Routine \tag{1}$$

A topic of interest is defined by a set of features or keywords extracted from the queries a user makes. Each topic has a relevance value associated with it that indicates how relevant the topic is for the user. In turn, each feature has an associated relevance value within a topic, which is given by the amount of times this feature was involved in a user's queries regarding that topic.

Formally, we can say that the set of *Topics* in a user profile is composed of n topics T_i with their associated relevance values $w(T_i)$:

$$Topics = \{(T_1, w(T_1)), (T_2, w(T_2)), \ldots, (T_n, w(T_n))\} \tag{2}$$

e.g. $Topics = \{(T_1, 0.8), (T_2, 0.6)\}$

In turn, a topic T_i is defined by a set of m features F_{ij} with their associated relevance values $w(F_{ij})$:

$$T_i = \{(F_{i1}, w(F_{i1})), (F_{i2}, w(F_{i2})), \ldots, (F_{im}, w(F_{im}))\} \tag{3}$$

Considering the user John Smith introduced in an example in Section 2, one of his topics of interest is composed of the following features:

$$T_1 = \{(department, 0.7), (product, 0.7), (sample - state, 0.8)\}$$

A feature F_{ij} may have one or more values v_{ijk}, with different weights $w(v_{ijk})$:

$$F_{ij} = \{(v_{ij1}, w(v_{ij1})), (v_{ij2}, w(v_{ij2})), \ldots, (v_{ijr}, w(v_{ijr}))\} \tag{4}$$

For example, the values that department can take are:

$$F_{11} = \{(storage, 0.7), (polymerization, 0.7)\}$$

Summarizing the ideas presented above and following the example, John Smith's profile comprises two topics, each defined by different sets of features:

$$Topics = \{(T_1, 0.8), (T_2, 0.6)\}$$
$$T_1 = \{(department, 0.7), (product, 0.7), (sample\text{-}state, 0.8)\}$$

$$
\begin{aligned}
F_{11} &= \{(storage, 0.7), (polymerization, 0.7)\} \\
F_{12} &= \{(propane, 0.7)\} \\
F_{13} &= \{(rejected, 0.85)\} \\
T_2 &= \{(department, 0.6), (product, 0.6), (sample\text{-}state, 0.8), \\
&\quad (sampling\text{-}point, 0.8)\} \\
F_{21} &= \{(arnipol, 0.7)\} \\
F_{22} &= \{(arnipol, 0.8)\} \\
F_{23} &= \{(A44, 0.36), (A46, 0.76), (A18, 0.33)\} \\
F_{24} &= \{(finished, 0.8)\}
\end{aligned}
$$

In our approach, a topic also includes the relationships (dependencies) among its feature values and the strength of these relationships. Thus, a feature value v_{ijk} of feature F_{ij} in topic T_i can be related to feature value v_{iqt} of feature F_{iq} in the same topic T_i. This relationship can be expressed by an arrow from v_{ijk} to v_{iqt}: $v_{ijk} \Rightarrow v_{iqt}$. Each relationship has a strength value associated with it given by the probability $p(v_{iqt}/v_{ijk})$, which conditions the presence of v_{iqt} in the topic to the presence of feature v_{ijk}. This situation is translated to queries by saying that the presence of v_{iqt} in a query depends on the presence of v_{ijk} in the query.

For example, the relationships between features in topic T_1 are the following (we omit the strength values for simplicity):

$$
\begin{aligned}
storage &\Rightarrow propane \\
polymerization &\Rightarrow propane \\
propane &\Rightarrow rejected
\end{aligned}
$$

A habit or routine is defined by temporal information describing the temporal categories to which a certain information need or topic is related. Each topic of interest can be associated with one or more routines. For example, consider that a topic of interest belonging to John Smith is: $T = \{$(department = *storage*, 0.7), (sample state = *rejected*, 0.8), (product = *propane*, 0.5)$\}$; and the routine associated with it is "early morning". This means that John Smith, when he gets to work every morning, makes a query to get information about rejected samples of propane originated at the *Storage* department.

More formally, a routine R is defined by three temporal categories: an hour category HC, a week category WC and a month category MC. The first indicates a period of hours within a day, the second the day of the week and the third the day number. For the example given in the previous paragraph $HC =$ "early morning", $WC = \oslash$ and $MC = \oslash$. The meaning of these categories will be explained in detail in Section 5.4.

A routine has a confidence value attached to it that indicates how sure the agent is about the user making queries on a given topic on the particular period of time represented in the routine. The confidence value is always computed with respect to a particular topic of interest.

Figure 3 shows an example of a database user profile. The following section describes in detail how each user profile component is obtained.

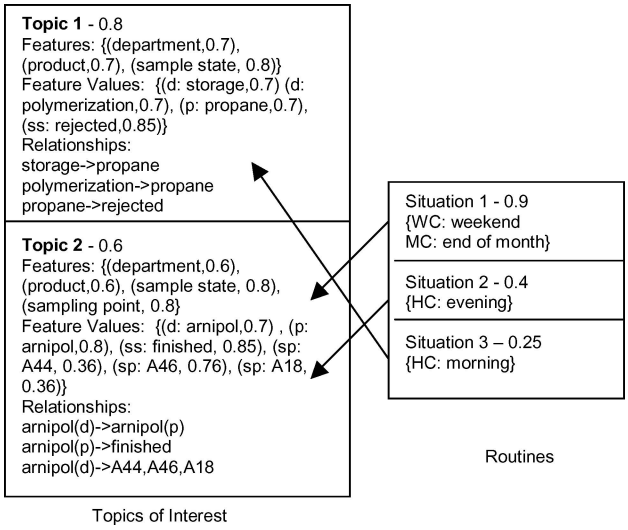

Figure 3. User profile.

5. Building and updating user profiles

To build a database user profile, *QueryGuesser* first observes a user's behavior while he is working with a database application and records data about each query the user makes. These data include the keywords involved in the query and temporal data indicating when the query was made. *QueryGuesser* uses the information recorded to cluster similar queries, since we consider that a set of similar queries represents a topic of interest. In turn, *QueryGuesser* determines the relationships among query features within a topic and the strength of these relationships. Once the topics are defined, the agent analyzes the queries to determine querying routines and to link routines to the corresponding topics.

This section is organized as follows. The observation task is explained in Section 5.1. The query clustering process is described in Section 5.2. Section 5.3 explains how the relationships among query features are obtained. Finally, Section 5.4 describes how the agent discovers a user's querying habits.

5.1. Observation of a user's behavior

To discover the different topics a user is interested in, *QueryGuesser* observes a user's behavior while he is working with an application through which he accesses the information stored in a database. This is generally done through an user-interface from which the user selects the features he is looking for, as the one shown in figure 4. These features are known as filters in the area of databases, and they are the values the user is looking for in certain columns in the tables that compose the database. In our example, John Smith made the query to obtain those *Propane* finished samples taken at *Warehouse A* at the *Storage* department

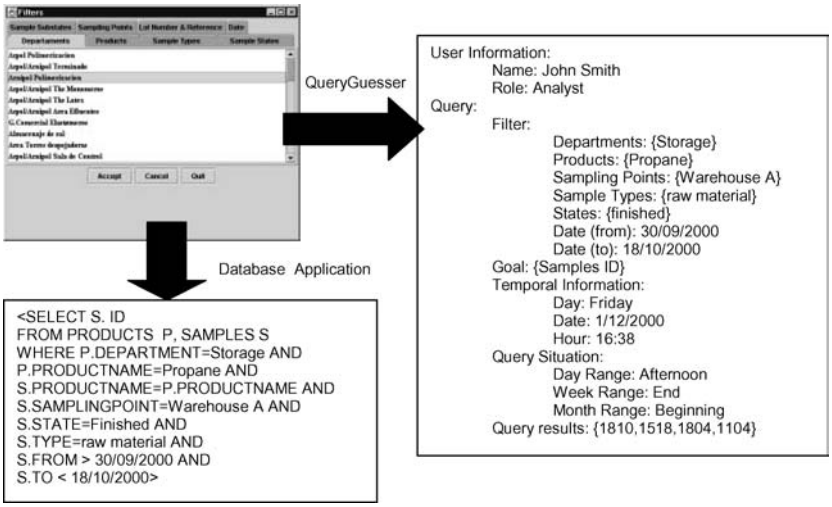

Figure 4. Data recorded from a query.

from 30th September, 2000 to 18th October, 2000. The application translates this into an SQL query, as shown at the left bottom of the figure, and the results the query returns are presented via a user-interface to the user, who can manipulate the information from there.

During observation *QueryGuesser* records data about each user query that enable it to discover both the user's topics of interest and his querying habits. The agent records: the filters used to make the query, i.e. the query features; the goal of the query; temporal information indicating when the query was made; and the query results. In the example shown in figure 4, the filter is composed of particular values of departments, products, sampling points, sample types, sample states, and sample extraction dates. The goals are the sample identification numbers that fulfill the conditions specified in the filter. The temporal data include the hour, the date and the day of the week when the query was executed. We can observe that the attributes appearing in the representation of the query correspond to the different conditions stated in the SQL query.

The information recorded during observation is used to build the user profile, as described in the following section.

5.2. *Query clustering*

In order to discover the different topics a user is interested in, a user's queries are clustered according to their similarity. We consider that a cluster grouping a set of similar queries constitutes a topic of interest. Thus, when data about a recent query are obtained, the agent compares the new query against the already recorded ones to determine the cluster to which it belongs. Each cluster is labelled by a cluster number, which is assigned to each query after categorizing it.

To achieve this goal, *QueryGuesser* compares the features that appear in the new query against the features that appear in the recorded queries. To be similar, two queries must contain the same or at least similar features. For example, a query involving departments and products is not similar to one containing sampling points and sample states. But a query involving departments, products and sampling points may be similar to one involving only departments and sampling points. In turn, the agent compares the values each feature takes in the two queries it is comparing. The agent needs a considerable percentage of the feature values to take the same value in the two queries in order to consider them as similar, as it is described below.

Similarity metrics are used to compare both the features involved in the queries and their values in order to determine the similitude between these queries. *QueryGuesser* uses a metric that computes a similarity function that gives as result a score representing how similar the queries are. If this score is higher than the threshold value τ specified by the metric, then the queries are similar and they share the same cluster. If not, a new cluster is created and the new cluster number is assigned to the query.

A domain expert establishes the degree of similarity between the different features that may be involved in queries. Feature matching is done by computing the distance of feature values in a qualitative scale. If two values belong to the same qualitative region, then they are considered as equal. Otherwise, the distance between their qualitative regions provides a measure of their matching score. In the case of features having lists of values, these lists should be identical, or one should contain the other or they should have values in common. Figure 5 shows two similar queries in the LIMS domain.

QueryGuesser assigns different importance values or weights to query features. The importance of a feature to a match depends on its overall impact to the comparison. The importance associated with each feature tells us how much attention to pay to matches and mismatches in that feature when computing how good a match is. For example, when

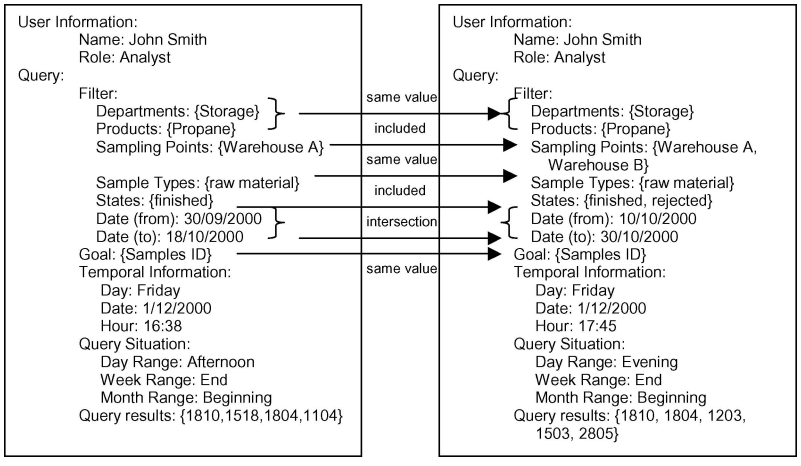

Figure 5. Matching of queries.

comparing two queries to determine if they are similar, the features used as filters and the goal of the query are highly important, but the features involving temporal aspects are not.

Equation (5) shows the function that *QueryGuesser* uses to compute the similarity between queries Q_N and Q_O. In that function, w_i is the weight of feature i, sim_i is the similarity function for feature i, and f_{iN} and f_{iO} are the values for features i in the new and old queries, respectively.

$$similarity(Q_N, Q_O) = \frac{\sum_{i=1}^{n} w_i * sim_i(f_{iN}, f_{iO})}{n} \tag{5}$$

For example, the calculus of the similarity between the two queries shown in figure 5 is shown in Eq. (6). The weight value associated with the attributes composing the filter and the goal is 1, and the weight associated with the rest of the attributes is 0. The function sim_i returns 1 if the values are equal, 0 if the values are different and a value between 0 and 1 if they are similar.

$$\begin{aligned} &\frac{sim_1(Storage, Storage) + \cdots + sim_7(SamplesID, SamplesID)}{7} \\ &= \frac{1 + 1 + 0.5 + 1 + 0.5 + 0.4 + 1}{7} = \frac{5.4}{7} = 0.77 \end{aligned} \tag{6}$$

QueryGuesser makes a ranking to determine the most similar query, where queries with higher scores are ranked higher than those with lower scores. The cluster number of the most similar query is assigned to the new one. Figure 6 shows an example of clusters grouping similar queries.

Regarding the user profile described in Section 4, the clustering of queries enables the agent to discover the different topics a user is interested in and to determine the features and feature values that each topic comprises. However, the agent cannot establish yet neither the relevance of the topics and features nor the relationships among features.

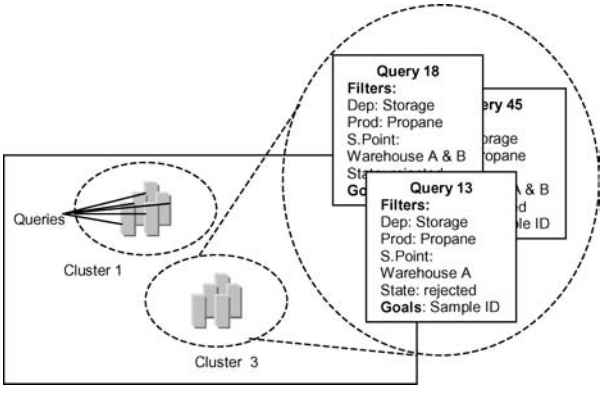

Figure 6. Clusters representing different topics of interest.

5.3. Managing relationships among query features

Once *QueryGuesser* has discovered via query clustering the different topics of interest a user has, it must determine the relevance each of these topics has for the user. Besides, it has to determine the relevance each feature has within a topic and the relationships among features. Our approach proposes the use of Bayesian networks to obtain this information. We have chosen this technique because it enables the agent to model both quantitative and qualitative information about the features composing the different topics a user is interested in. Besides, Bayesian inference mechanisms enable us to make inferences about the queries a user can possibly make. Using Bayesian networks is just a viable approach; other machine learning techniques could have been also used. Our goal was giving a solution to the problem of learning a database user's profile, we did not mean neither to find the best solution nor to compare different approaches.

In the last decade, interest has been growing steadily in the application of Bayesian representations and inference methods for modeling the goals and needs of users (Horvitz, 1997). A BN is a compact, expressive representation of uncertain relationships among parameters in a domain (D'Ambrosio, 1999). A BN is a directed acyclic graph that represents a probability distribution where nodes represent random variables and arcs represent probabilistic correlation between variables. For each node, a probability table specifies the probability of each possible state of the node given each possible combination of states of its parents. Tables for root nodes just contain unconditional probabilities (Haddawy, 1999).

In our work BN are used to model, both qualitatively and quantitatively, the topics of interest of a given user. A node represents a feature or keyword used by the user to make a query. Each variable can have only two values: true, representing that the item is present in a user query, and false, indicating that the item is absent. Arcs model existing relationships between features in the given domain. Probability values indicate the strength of these relationships, and they represent how the presence of a feature in a query influences the presence of other feature in the query. Section 5.3.1 explains how BN are built.

Once the agent has constructed a BN that models the dependencies among features belonging to a user's topics of interests, it applies Bayesian inference mechanisms to discover the relevance values each feature has within the topics and the relevance each topic has for the user. This process is explained in Section 5.3.2.

5.3.1. Building the BN. *QueryGuesser* requires that a domain expert establishes the parameters it needs to build the BN: the features that users can use as filters to make queries, the values each feature can take and the dependencies that exist between them. The features involved in queries are the column names of the tables composing the database, and the values these columns take are the feature values. For example, the *Sample* table in the LIMS database has 6 columns: sample-id, product, department, state, sampling point and date. All these columns are features that can be used in a user's queries and, in consequence, they are potential nodes in the user's BN. As regards values, the product feature for instance, takes as many values as products exist in the petrochemical plant.

QueryGuesser requires that a domain expert establishes the dependencies among variables in the domain. This can be a drawback in those cases where a domain expert is not

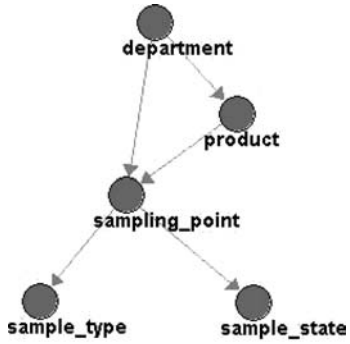

Figure 7. Generic relationships in the LIMS domain.

found. If this is the case or the relationships are incomplete, the network structure could be obtained using Bayesian learning algorithms, such as the ones described in Heckerman (1999) and Jensen (1996). A log of queries will be used in this case as an input for the learning algorithm in order to discover the dependencies among query features. The problem with these algorithms is that, sometimes, some existing relationships are not found.

In the LIMS domain, we can point out some relationships that exist between query features. Each employee has access to information about samples taken or originated at a set of departments or sections within the petrochemical plant. Each department is in charge of tracking samples belonging to a set of products and taken at certain sampling points. Each product can be sampled at certain sampling points within the plant. The types of samples taken at a department depend on the characteristics of the sampling point. Besides, a technician can make queries about particular states of samples depending on the sampling point, e.g. rejected samples in conflicting sampling points. These restrictions are modeled using BN by establishing relationships between the corresponding nodes. These relationships model the probability that a child node (e.g. a sample state) is used in a query given that a parent node is used in the query too (e.g. a sampling point). Figure 7 shows these "generic" relationships in the LIMS domain.

The BN of a given user is gradually instantiated as he makes queries to the database. When a user makes a query, one node is added to the network for each new feature appearing in the query. Arcs are drawn between the corresponding nodes considering the generic relationships established for the particular domain. Each feature is also linked to the code node of the topic (cluster) to which it belongs. In this way, the cluster node is linked to all its constituents.

Probability values are updated as the frequencies of features in queries are modified with each new query. This kind of learning is known as sequential learning (Friedman and Goldszmidt, 1997; Spiegelhalter and Lauritzen, 1990) since we sequentially adapt the network parameters and the network structure as new information is obtained.

Figure 8 shows an example of a BN that models John Smith's interests in the LIMS domain. This network is the result of a sequence of queries made by the user when he was looking for information about samples. The model shown in the figure is completed by

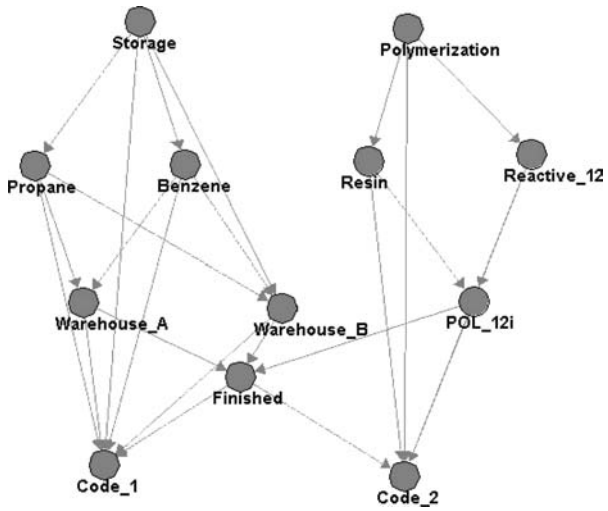

Figure 8. BN representing John Smith's interests.

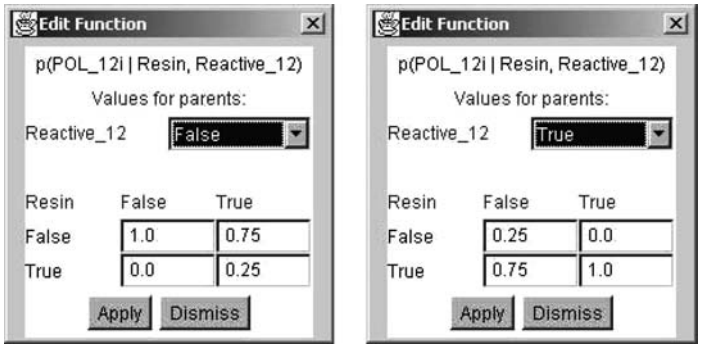

Figure 9. CPT associated with the POL 12 node.

establishing the probability values in the form of conditional probability tables (CPTs). The tables in figure 9 represent the CPT associated with the *POL 12* node. This table shows the probability values the *POL 12* node takes given a combination of the states of the *Resin* and *Reactive 12* nodes. *POL 12* is a sampling point, while *Resin* and *Reactive 12* are products. This CPT indicates the influence the presence (or absence) of these products in a query has on the presence (or absence) of the sampling point in the query.

5.3.2. Applying bayesian inference mechanisms. Once *QueryGuesser* has built a BN for a user, it uses Bayesian inference mechanisms to discover the relevance of topics and the relevance of topic components. There are different types of inference mechanisms: causal or top-down inference to obtain a probability value for a variable having as evidence an

ancestor; and diagnostic or bottom-up inference to obtain a probability value for a variable having as evidence a descendant (Nilsson, 1998). Our approach uses both types of inference. *QueryGuesser* uses causal inference to determine the feature values relevant to a given user and diagnostic inference to discover the relevance of feature values within a given topic of interest. From the various existing algorithms developed to perform inference in BN, we use the bucket elimination algorithm (Dechter, 1996; Cozman, 2000).

To determine the components of the different topics of interest, we have to distinguish among different types of features. Some features are independent, i.e. their nodes do not have arcs coming from other nodes in the BN. On the other hand, some types of features depend on others, i.e. they have one or more features as parents in the BN. *QueryGuesser* determines: relevant independent features, relevant dependent features, relevant values for each relevant feature, relevant values of a feature given values of other features, combinations of relevant feature values, and relevant feature values given a topic of interest. An independent feature value is relevant if its simple probability value is higher than a specified threshold λ considered as the minimum level of relevance required by the agent. To determine the importance of a dependent feature value, our agent first sets as evidence the relevant values of its parents. Then, via Bayesian inference mechanisms, the agent determines which values of the child feature have higher posterior probability values. For example, considering the network in figure 8 the relevance of the department nodes is, directly, the simple probability value associated with them. Suppose that the relevance of *Storage* is 0.83 (i.e. the probability that *Storage* appears in a query is 83%) and the relevance of *Polymerization* is 0.33 (i.e. the probability that *Polymerization* appears in a query is 33%). If we consider a threshold value of $\lambda = 0.40$, then the only relevant department is *Storage*. The relevance of dependent features, such as products, sampling points and states, is computed using as evidence the values of their parents. For example, to find out the relevant products, we set as evidence a true value for *Storage* and then we compute the posterior probabilities of each children. Those obtaining probability values higher than the specified threshold are considered as relevant.

The next step is determining the relevance value associated with each relevant topic. Topics are represented in the Bayesian model through code nodes (cluster numbers). Relevance values are inferred by setting as evidence the relevant inferred values for each feature belonging to the topic involved, and computing then its posterior distribution. Finally, a ranking with the different topics is built, considering as relevance values those obtained via inferences.

As a result of the process described above, *QueryGuesser* has completed the definition of a user's topics of interest. Now, the agent has learned the relevance of each topic, the relevance of each feature within a topic, the relationships that exist among topic features and the strength of these relationships.

5.4. Discovering querying habits

To detect some behavioral patterns in a user's queries, it is necessary to record information about the moments or situations in which this user makes queries to the database. The *QueryGuesser* agent considers different temporal categories or periods within a day, a

Table 1. Hour categories.

Hour category	Hours involved
Early morning	5–8
Morning	8–12
Midday	12–14
Afternoon	14–17
Evening	17–20
Night	20–22
Midnight	22–24
Dawn	0–5

Table 2. Day categories.

Day category	Days involved
Beginning	Monday, Tuesday
Middle	Wednesday, Thursday
End	Friday, Saturday, Sunday

Table 3. Month categories.

Month category	Days involved
Beginning	1 to 10
End	26 to 31
Default	The rest of the days

week and a month, and it associates the appropriate periods with each query. Tables 1 to 3 show the different categories the agent uses to categorize queries according to their date and time. For the example in figure 4 in Section 5.1 these periods are: afternoon (hour category), end (for the week category) and beginning (for the month).

QueryGuesser detects a user's behavioral patterns from the information stored during observation: by comparing the temporal categories assigned to two different queries the agent determines whether these queries are commonly executed or not in similar situations. The comparison is done regarding the proximity of situations both in qualitative and in quantitative scales. For example, when comparing the hour categories of two queries, they are similar if they have the same value or if their categories are neighbors (e.g. early morning and morning). When comparing the week and the month categories their values have to be necessary equal for the routines to be similar.

To determine the most frequent temporal categories we use an heuristic based on the frequency of occurrence of recorded queries. The agent obtains the different "situations" in which a query was executed and it determines the most frequent hour category. Then,

it determines the most frequent day category within a week, and finally it determines the most frequent day category within a month. Complete situations involving combinations of the three types of categories are also analyzed.

By varying the matching functions that compare queries, we can set the similarity level we want to obtain when detecting routines. For example, for some purpose a query made on the 2nd day of every month is similar to those made on the 10th, because we are considering the beginning of the month. However, these situations can be considered different for some other purposes.

The routine detection process enables *QueryGuesser* to discover the routines attached to the different topics of interest. This means that the agent has discovered when the user makes queries belonging to each topic of interest.

5.5. *Adjusting the profile with user feedback*

The *QueryGuesser* agent becomes more competent as its knowledge about a user's interests and habits is augmented and refined as it interacts with the user. A user can provide feedback for the suggested queries and inferred habits. The user feedback causes modifications to a user profile by adjusting the relevance value associated with each topic and with the features that compose each topic of interest for this user. There are two types of feedback: implicit or indirect, and explicit or direct. The user provides explicit feedback when, for example, he gives an evaluation for a query suggested by the agent. The user can rank a suggested query or a notified new item as relevant, more or less relevant, or irrelevant to his information needs. The implicit feedback has to be obtained via the observation of a user's actions once the agent has made its suggestions. Actions such as making new queries, making the queries proposed by the agent, getting query results, ignoring the suggested new piece of information, and ignoring suggestions are considered. By adjusting the profile according to the user feedback, *QueryGuesser* adapts itself to the user's changing interests and provides him assistance accordingly. The profile is adjusted by including the user feedback in the calculation of the relevance associated with each topic of interest and with each feature describing them.

We use the formula shown in Eq. (7) to compute the relevance value Rel of a query feature i. This formula is an adaptation of Rocchio's formula (Rocchio, 1971), which is commonly used in Information Retrieval to consider relevance feedback in document weight calculus. In our formula α, β and γ are the weights of the different terms in the equation (we use $\alpha = 0.8$, $\beta = 0.15$ and $\gamma = 0.05$), $\mathrm{Rel}_i^{\mathrm{old}}$ is the relevance value of feature i obtained from the BN, PF_i is the amount of positive feedback, NF_i is the amount of negative feedback and TF_i is the total amount of feedback provided by the user. A similar formula is used to compute the relevance of a topic.

$$Rel_i = \alpha * Rel_i^{old} + \beta * \frac{PF_i}{TF_i} - \gamma \frac{NF_i}{TF_i} \tag{7}$$

For example, consider the feature *Storage* whose initial relevance value was 0.83. Suppose that the agent suggested a query involving *Topic 1* to which this feature belongs, and

that the user ranked this query as relevant. Thus, the new relevance value is computed as follows:

$$Rel_{Storage} = 0.8 * 0.83 + 0.15 * \frac{1}{1} - 0.05 * \frac{0}{1} = 0.814 \tag{8}$$

6. Experimental results

We implemented the *QueryGuesser* agent in Java and we used JavaBayes[3] to manipulate BN. The chosen programming language makes our agent independent of a particular database technology or platform. The *QueryGuesser* agent was tested while assisting the users that operate the subsystem in charge of sample tracking within a LIMS in a petrochemical plant. The main goal of these tests was to analyze if the content of user profiles improved as the agent interacts with the user, tending to the user's needs. We studied the effects of user feedback in the profile building process, since in this way we can view the evolution of user profiles (Sheth and Maes, 1993). We used the following parameters to perform the tests: $\tau = 0.60$ and $\lambda = 0.40$. These values were selected after performing a series of experiments to determine the most appropriate threshold values.

In order to make the tests we developed a series of controlled experiments with 20 (twenty) users. Each of them made 30 (thirty) queries and the agent provided assistance to them 7 (seven) times (at different time moments for the different users). We chose this methodology because controlled experiments enable us to settle the context we need to test our agent (consistency between users' actions and their interests, explicit feedback, among others). It would take as long achieving this in an uncontrolled environment.

When users were working with the database application, the agent provided assistance to them. Once the agent has assisted the user, we asked each user to provide feedback for each query suggested by the agent, rating them as relevant or irrelevant. The graph in figure 10 plots the performance of the *QueryGuesser* agent. For each assistance session (i.e. each time the agent provides assistance to the user), we recorded: the number of queries suggested by the agent, the number of suggested queries that were relevant to the user and the number of queries that were not relevant to him. The graph plots, for each session, the average

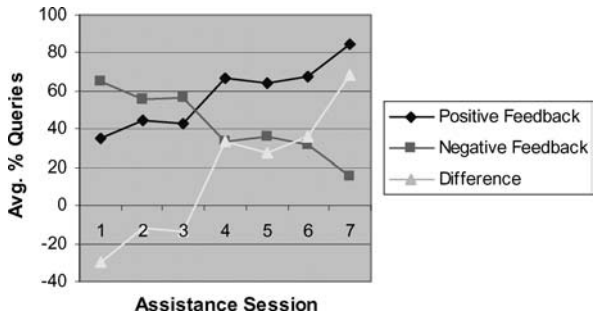

Figure 10. Evolution of user profiles.

percentage of suggested queries that received positive feedback, the average percentage of suggested queries that received negative feedback and the difference between them. We plot the proportion of queries instead of the absolute numbers of queries because the numbers of queries suggested for each user varies according to their interests.

We can observe that there is a pattern of improving performance in the queries suggested by the agent. We can observe a soft learning curve in the initial states, since the first feedback values are mostly negative. This situation arose because we did not wait until the agent had acquired a considerable amount of knowledge about users to start testing it. In this way, the agent gave assistance to the user having information only about a few queries made by the users. But then, due to the users' feedback, the agent stopped suggesting too general (irrelevant) queries and feedback values became positive. The number of suggested queries that received a positive feedback give us an indication of the amount of user work the agent saved, since the agent made these queries instead of the user. The agent also "thought" of making them, not the user, because the agent anticipated the user's needs.

The experiments we carried out demonstrated that our approach improved the user's interaction with the database application since *QueryGuesser* provided them effective assistance. This is derived from the improvement in the accuracy of user profiles and from the increase of positive user feedback.

7. Related work

Interface agents have been used in a wide variety of domains such as information filtering and information retrieval (Billsus and Pazzani, 1999; Cordero et al., 1999), e-commerce (Morris et al., 2000), web browsing assistance (Lieberman et al., 2001; Godoy and Amandi, 2000), e-mail management (Boone, 1998; Segal and Kephart, 2000), e-learning (Amandi et al., 2003) and meeting scheduling (Mitchell et al., 1994), among others. Our agent is innovative in the sense that interface agent technology has not been applied to database systems and LIMS, as far as we know.

Different interface agents assisting users in domains such as those mentioned above, use different machine learning techniques to build user profiles. For example, the *PersonalSearcher* agent uses hierarchical clustering to obtain the different Web topics a user is interested in Godoy and Amandi (2000); *NewsAgent* uses case-based reasoning and a topic hierarchy to discover which newspaper news a user prefers (Cordero et al., 1999); *SwiftFile* uses a TF-IDF style classifier to organize emails (Segal and Kephart, 2000); and *CAP* uses decision trees to learn users' scheduling preferences (Mitchell et al., 1994).

The selection of the technique depends on the task the agent has to perform and on the application domain. We have chosen BN and clustering to build our profiles. Some other projects have used BN in domains with inherent uncertainty (Horvitz et al., 1998; Lee et al., 2001; Conati et al., 2002). For example, the Lumiere project at Microsoft Research (Horvitz et al., 1998) uses BN to infer a user's needs by considering a user's background, actions and queries (help requests). Based on the beliefs of a user's needs and the utility theory of influence diagrams (an extension to BN), an automated assistant provides help for users. This project ended up in the well known Office Assistant first incorporated into Microsoft Excel. The problem with this assistant is that the BN is static and it is not updated as the user

interacts with the system. Thus, the assistant has no learning capabilities and sometimes the model does not match the user's profile.

There are some works that analyze a user's queries in different contexts with various purposes. These works have focused on learning a user profile from a user's queries, but in domains different from databases. For example, (Lenz et al., 1998; Burke et al., 1997) use Case-Based Reasoning (CBR) to answer queries in help desk applications (FAQs, frequently asked questions). Lau and Horvitz (1999), Wen et al. (2002), Beeferman and Berger (2000) try to infer a user's patterns of search in Web search engines by analyzing query logs. Lau and Horvitz (1999) discusses the construction of probabilistic models (dynamic BN) centering on temporal patterns of query refinement in search engines. The approach presented in Wen et al. (2001, 2002) clusters similar queries according to their contents for FAQs identification. DBSCAN clustering algorithm is used. The approaches used in these works have been analyzed and discarded. For example, CBR cannot be applied since although queries are similar the results obtained from the database may vary instead of keeping equal as in FAQs. The DBSCAN clustering algorithm is not suitable for our domain since many of the tasks it involves are not necessary (e.g. stemming, stop word removal, phrase recognition).

8. Conclusions

In this paper, we have presented *QueryGuesser*, an agent developed to assist users in their work with a database system. Our agent has been successfully tested in a LIMS, assisting users who make queries from distant places in a petrochemical plant. These users had considerable delays in getting the information they needed and their work was quite routine. Thus, our agent was very useful to them. The experimental results obtained so far have proved that our agent is good (and promising) at determining a database user's topics of interest.

We can point out two main contributions of our work, one within the area of applications that use databases intensively and the other within the area of user profiling in agent development. First, using interface agents to assist users in their work with a database system is a novel application. The *QueryGuesser* agent assists such users by personalizing their query making process according to their interests and habits. Current database management systems do not offer such services to users. Second, we combined clustering and BN to build user profiles. BN allowed us to model the relationships among query features both in a qualitative and in a quantitative way. Bayesian inference mechanisms enabled us to build a user's profile based on the queries we recorded and clustered. By clustering similar queries we could analyze the different topics of interest a user has. We could not have done it with other approaches, such as considering single queries and ordering them by frequency of occurrence, for example.

Acknowledgments

We would like to thank Analyte company for providing us the necessary media to test our agent in a LIMS domain.

Notes

1. One who uses a program that was specially made to suit him or his needs.
2. For simplicity, we use "she" for the assistant and "he" for the user, but we do not mean to be sexist.
3. http://www.cs.cmu.edu/~javabayes/index.html

References

Amandi, A., Campo, M., Armentano, M., and Berdún, L. (2003). Intelligent Agents for Distance Learning. *Informatics in Education* 2(2), 161–180.

Beeferman, D. and Berger, A. (2000). Agglomerative Clustering of a Search Engine Query Log. In: *KDD 2000* (pp. 407–416).

Billsus, D. and Pazzani, M.J. (1999). A personal news agent that talks, learns and explains. In O. Etzioni, J.P. Müller, and J.M. Bradshaw (Eds.), *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*. ACM Press: Seattle, WA, USA (pp. 268–275).

Bonett, M. (2001). Personalization of Web Services: Opportunities and Challenges. *Ariadne* Issue 28.

Boone, G. (1998). Concept Features in Re:Agent, an Intelligent Email Agent. In *Proceedings of the Second Interntational Conference on Autonomous Agents—Agents 98* (pp. 141–148).

Burke, R., Hammond, K., Kulyukin, V., Lytinen, S., Tomuro, N., and Schoenberg, S. (1997). Question Answering from Frequently Asked Question Files. *AI Magazine*, 18(2), 57–66.

Conati, C., Gertner, A., and VanLehn, K. (2002). Using Bayesian Networks to Manage Uncertainty in Student Modeling. *Journal of User Modeling and User-Adapted Interaction*, 12(4), 371–417.

Cordero, D., Roldán, P., Schiaffino, S., and Amandi, A. 1999. Intelligent Agents Generating Personal Newspapers. In *Proceedings ICEIS 99, International Conference on Enterprise Information Systems* (pp. 195–202).

Cozman, F. (2000). Generalizing Variable Elimination in Bayesian Networks. In *Workshop on Probabilistic Reasoning in AI—IBERAMIA-SBIA 2000* (pp. 27–32).

D'Ambrosio, B. (1999). Inference in Bayesian Networks. *AI Magazine*, 20(2), 21–35.

Dechter, R. (1996). Bucket Elimination: A Unifying Framework for Probabilistic Inference. In *12th Conf. On Uncertainty in AI* (pp. 211–219).

Friedman, N. and Goldszmidt, M. (1997). Sequential Update of Bayesian Network Structure. In *Thirteenth Conference on Uncertainty in Artificial Intelligence* (pp. 165–174).

Godoy, D. and Amandi, A. (2000). PersonalSearcher: An Intelligent Agent for Searching Web Pages. *Advances in Artificial Intelligence—Lectures Notes in Artificial Intelligence, LNAI 1952* (pp. 43–52).

Haddawy, P. (1999). An Overview of Some Recent Developments in Bayesian Problem-Solving Techniques. *AI Magazine* 20(2), 11–19.

Han, Y. and Sterling, L. (1997). Agents for Citation Finding on the World Wide Web. In *PAAM 97* (pp. 303–318).

Heckerman, D. (1999). A Tutorial on Learning with Bayesian Networks. *Learning in Graphical Models* (Also appears as Technical Report MSR-TR-95-06, Microsoft Research, March, 1995).

Horvitz, E. (1997). Agents with Beliefs: Reflections on Bayesian Methods for User Modeling. Invited Talk at *6th International Conference on User Modeling*.

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. In *Fourteenth Conference on Uncertainty in Artificial Intelligence* (pp. 256–265).

Jensen, F.V. (1996). *An Introduction to Bayesian Networks*, Springer Verlag: New York.

Lau, T. and Horvitz, E. (1999). Patterns of Search: Analyzing and Modeling Web Query Refinement. In *Proceeding 7th International Conference On User Modeling* (pp. 119–128).

Lee, S.-I., Sung, C., and Cho, S.-B. (2001). An Effective Conversational Agent with User Modeling Based on Bayesian Network. *Web Intelligence: Research and Development: First Asia-Pacific Conference, WI 2001—Lecture Notes in Computer Sciences 2198* (pp. 428–432).

Lenz, M., Hübner, A., and Kunze, M. (1998). Question Answering with Textual CBR. In *3rd International Conference (FQAS' 98)*, Vol. 1495 of *Lecture Notes in Computer Science* (pp. 236–247), Springer-Verlag.

Lieberman, H., Fry, C., and Weitzman, L. (2001). Exploring the Web with Reconnaissance Agents. *Communications of the ACM*, 44(8), 475–484.

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

Mitchell, T., Caruana, R., Dermott, J.M., and Zabowski, D. (1994). Experience with a Learning Personal Assistant. *Communications of the ACM* 37(7), 80–91.

Morris, J., Ree, P., and Maes, P. (2000). Sardine: Dynamic Seller Strategies in an Auction Marketplace. In *Proceedings 2nd ACM Conference on Electronic Commerce EC 00* (pp. 128–134).

Nilsson, N. (1998). *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann.

Rocchio, J. (1971). Relevance Feedback in Information Retrieval. *The SMART Retrieval System: Experiments in Automatic Document Processing—Chapter 4* (pp. 313–323).

Segal, R. and Kephart, J. (2000). Swiftfile: An intelligent assistant for organizing e-mail. In *In AAAI 2000 Spring Symposium on Adaptive User Interfaces*.

Sheth, B. and Maes, P. (1993). Evolving Agents for Personalized Information Filtering. In *Proceedings 9th IEEE Conference on Artificial Intelligence for Applications (CAIA 93)* (pp. 345–352).

Spiegelhalter, D.J. and Lauritzen, S.L. (1990). Sequential updating of conditional probabilities on directed graphical structures. *Networks* 20, 579–605.

Wen, J., Nie, J., and Zhang, H. (2002). Query Clustering Using User Logs. *ACM Transactions on Information Systems*, 20(1), 59–81.

Wen, J.-R., Nie, J.-Y., and Zhang, H.-J. (2001). Clustering user queries of a search engine. In *World Wide Web* (pp. 162–168).