TRIP@DVICE MOBILE EXTENSION OF A CASE-BASED TRAVEL RECOMMENDER SYSTEM

Quang Nhat Nguyen, Dario Cavada and Francesco Ricci

eCommerce and Tourism Research Laboratory, ITC-irst, via Solteri 38, 38100 Trento, Italy

{quang,dacavada,ricci}@itc.it

ABSTRACT

In the majority of e-commerce applications for travel and tourism, users usually experience problems related to an overload of information and options to consider. Often the user receives a poor support in finding, filtering, comparing, and combining products. For the on-the-move traveller, finding interesting travel products is often highly expensive (time, and cognitive effort). We propose an interactive, case-based, and judgment-based approach to the problem of mapping travellers' needs to the available travel products and services. The approach is case-based since it uses the knowledge of past recommendations when making a new one. The approach is based on an interactive adaptation of the dialogue since it involves the user in the process of searching products. The approach is judgment-based since it allows the user to provide feedbacks to the system's recommendations, and incorporates such feedbacks in the successive recommendations. The user is not required to define a completely specified set of needs and wants at the beginning, but is guided through a recommendation process to interleave needs elicitation with recommendations. This approach has been implemented in a prototype which is here described.

1 INTRODUCTION

eCommerce tourist web sites allow the user to search in catalogues for those products that match the user's preferences and constraints. This simple approach has some limitations when applied to the tourism domain. In fact, tourism products have a complex structure, whose definitions have not been standardized (Werthner and Klein, 1999). Moreover, the tourist decision process is much more complex than that typically required for selecting a CD or a movie (Fesenmaier and Jeng, 2000). Basically, the user's preferences cannot be easily translated into product features and finally in recommendations. More precisely, the recommended destinations should vary according to the nature of each trip and user characteristics. The recommendations should reflect personal differences in the nature of the information sought and processing styles. A destination recommender system should facilitate experiential aspects of the decision and information searching process (Hwang, Gretzel and Fesenmaier 2002).

Nowadays in a large number of web sites related to travel and tourism, users usually have problems due to an overload of information and options to consider, since they often receive a poor support in finding, filtering, comparing, and combining products. Moreover, they may not have enough knowledge to formulate and reformulate the query in order to retrieve some good results. For the on-the-move traveller, finding interesting products may be very difficult in terms of cognitive effort, time and connection cost.

The main objective of our research is to design and develop software agents that support the full travel life cycle, from pre-travel planning, through on-tour (i.e., on-the-move) support, and finally post-travel support. For the pre-travel stage we have developed NutKing, a recommender system that combines content-based and collaborative-based filtering methods to build the recommendation. NutKing is developed using recommendation technologies that are included in

Trip@dvice, a product that our laboratory is going to commercialise. In the content-based filtering phase, NutKing supports the user in searching the product catalogues. Moreover, in cases of failure situations (e.g., too many results or no result found), NutKing provides some query refinement suggestions (i.e. constraints that could be discarded or added) in order to help the user to find a suitable number of products.

In principle, travel planning is a perfect application for a mobile user. In fact, new mobile devices have more and more power in terms of CPU, memory, display size, and application programming language. The new generation wireless networks (e.g., GPRS and UMTS) have introduced new tools to better present information (minor effort from user) and to let the user access quicker the right information chunk (less cost and less time spent on). However, mobile devices still suffer from some restrictions compared to real web based systems. Moreover the mobile context imposes a quite different interaction management and decision model so NutKing is not directly usable on a mobile device. Thus we have developed a completely new application, called Mobile Intelligent Travel Recommender (mITR), that helps the traveller to complement his travel plan (e.g., add new events at the destination). These complementary products should conform to what have already been selected in the pre-travel stage; and also the time and user effort to find them should be minimized. Because of the inherent limitations of mobile devices (e.g., small screens, low speed processors, etc.), users really need the system's support in finding travel information of his needs and in selecting travel products of his interest.

The important objective of mITR is to improve iteratively, on each recommendation step, the fitness of the recommended products to user needs. mITR achieves this objective by taking into account user feedbacks collected during the interaction. Instead of asking the user to input explicitly his needs, mITR exploits the knowledge contained in the pre-travel plan to guess an initial set of candidate products. The user is then required to select one of the recommended products or to provide critics (i.e., feedbacks) to some of these products, such as: "I like this feature of this product", "I want something less expensive", "I dislike this feature of this product". The system uses those feedbacks to update the query and re-run the selection and ranking algorithm implemented in Trip@dvise. Our approach is inspired by some previous approaches (Shimazu, 2001; Ginty et al., 2002; Burke, 2000) that have introduced the ideas of "recommendation by proposing" and "similarity based query revision". The originality of our approach resides in the combination of logical and similarity-based queries and in the method used to identify the initial set of candidate products that exploits the case base of travel plans contained in the NutKing system. Moreover, none of the previous approaches are implemented on a mobile device.

In the post travel phase, the user can give his feedbacks on the recommended products. Such feedbacks could be used to better understand the user's needs. The traveller may also access the system to receive some customer support and offers (e.g., a bonus/gift for frequent travellers), to recommend a travel product/package to one of his friends, or to upload some photos taken with the mobile phone to share with other travellers.

We start explaining the NutKing's main interactions and the use of pre-travel information for improving the mobile recommendation. Then we talk about the methodology used in mITR to reduce the dialog length. Finally, we will see an example of mITR recommendation process and the technology used.

2 TRIP PLANNING WITH TRIP@DVICE

This section describes a typical user/system interaction and shows how NutKing can be used as a pre-travel planning tool. It supports the interaction mimicking a typical counselling session in a real travel agency. The user initially defines the major trip goals and constraints by entering some

preferences as shown in Figure 1. These are general features of the travel, like travel companions, budget and transportation means. These wishes are used both to recommend products, exploiting other users' trips, and to set some default constraints in successive query forms. Furthermore these wishes will be used in the mITR for constructing the initial constraints and for ranking the initial query result. After this initial step, the user can start bundling his trip by searching for travel products, whatever he wants. This could be a destination or an accommodation or an event or an attraction (in the available catalogues).

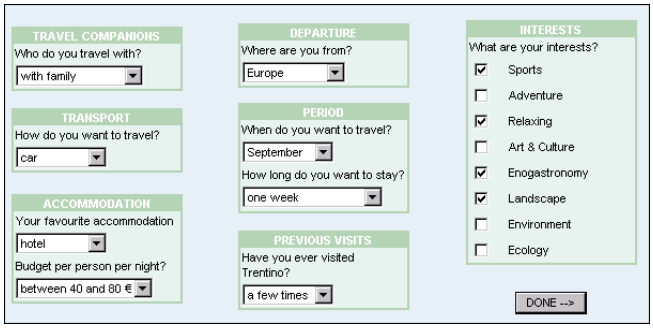

Figure 1: General travel wishes

Let's assume that the user is looking for an accommodation; the system offers to the user a classical Query-By-Example interface, where he can set constraints on the product features. If the query fails because no products satisfy the query, the system proposes some alternatives of query relaxations that, if applied, would provide to the user a suitable result set.

Another possible situation may occur when the user selects too few constraints and the query retrieves too many products to be examined. In this case NutKing asks the user to provide some additional (alternative) constraints.

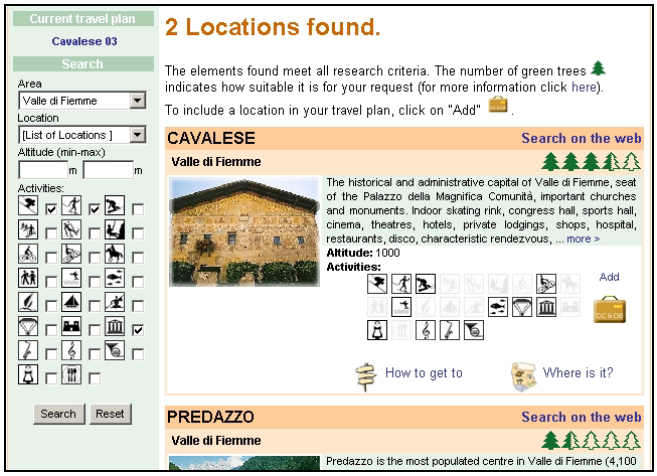

Figure 2: List of recommended products

At the end of this interaction a list of recommended products are shown (Figure 2). All of these satisfy the (explicit) user constraints, but furthermore are ranked first the product most similar to those selected by other users having expressed similar general wishes. A similar approach is used in mITR in order to retrieve the first list of best suitable products.

Then the user can add the selected product to his travel bag and proceed by adding other products to his trip. Finally, the user can print the complete travel in a brochure-like style and he can download some information on the mobile device (e.g., the address and the phone number of the hotel).

3 TRIP@DVICE MOBILE EXTENSION

When an on-the-move traveller starts searching for a desired travel product, the mITR system constructs, and iteratively updates, a query which represents the traveller's needs. User needs are collected during the interaction as they are required to further proceed. Therefore, the amount of the user's input is minimized. Queries are processed by the system to produce a set of recommended products that are presented to the user at the end of each iteration step (a recommendation cycle).

Hence, the problem of supporting the user in selecting a travel product is equivalent to the problem of building the right query; that is, the one that better represents the user's needs. In this section, we firstly introduce the approach to the problem of updating the query, then illustrate this approach with an example of recommending restaurants, and finally discuss about the software technology used.

3.1 Query update in mITR

The query, that represents the user's needs, is built using conditions exploited from the pre-travel plan and the current context (e.g., location and time). The query contains two components:

- the logical component of the query, Q_L, contains some logical constraints that must be necessarily satisfied;
- the similarity-based component, Q_S, contains a product pattern (i.e., a partially defined product) that is computed using products found in other travel plans stored in NutKing and similar to the current user's travel plan.

Therefore, the logical query component contains strict constraints (mainly related to the context) whereas the similarity-based one contains soft constraints (dealing with optional personalization issues). We denote this combination with $(Q_L|Q_S)$.

The query $(Q_L|Q_S)$ is initialised when the user starts searching for a travel product. The logical query (Q_L) is initialized with contextual constraints. We impose a constraint on the location, assuming that the user is interested in products available in a neighbourhood of his/her current location. Moreover, we impose that these products be available at the time when the request is made.

The similarity-based query (Q_S) is initialised using the knowledge exploited from the past cases (the past recommendations) of the NutKing system. If the user has registered to the NutKing system, and already requested (in the past) some recommendations on the same type of travel product that he is searching (e.g., restaurants), then these previously recommended products (i.e., the previously recommended restaurants) are exploited to construct the query Q_S. If the user has registered, but never asked for such type of travel product (e.g., restaurants), then the similarity-based query is constructed using the previously recommended products (of this type) of the other users who have expressed needs similar to those contained in the pre-travel plan of the user. If the user has not registered to the NutKing system, the query Q_S is initialized as an empty pattern; i.e., all features of this empty pattern are unknown.

In order to produce the ranked list of candidate products, first the logical query (Q_L) is executed to return the constraints-matching products; that is, the set of products which satisfy all constraints in the logical query. Then, all the constraints-matching products obtained are scored and ranked corresponding to their similarity with the similarity-based query (Q_S). The similarity computation returns an ordered set of candidate products (i.e., the top five of the set of constraints-matching products). Finally, all candidate products are shown to the user as the current recommendation.

Before describing the query updating process and the types of user feedbacks supported by the system, let us present some formal definitions:

- Products are represented as vectors of feature values $x=(x_1,\ldots,x_n)$. The feature value x_i may be numerical, nominal, symbol set, or text. In the rest of the paper, we will illustrate the system functionality using an example of restaurant recommendation. For the sake of simplicity, here we will represent a restaurant with five features: name (nominal), type (nominal), location (nominal), cost (numerical), and openingDays (symbol set). Hence, the example ("Maso Burba", "spaghetteria", "Trento", 10, {3,4}) means that the name is x_1="Maso Burba", the type is x_2="spaghetteria", the location is x_3="Trento", the cost is x_4=10, and the restaurant is open on Tuesday and Wednesday x_5={3,4}.
- The logical query in principle could be written in SQL but we consider a simpler language, to allow possible query refinement. Query refinement for a general SQL query is an intractable problem. Hence, in our system, a logical query is constructed by the conjunction of constraints; each of them dealing with only one feature, and a feature appears in only one constraint.

  $$Q_L = c_1 \wedge \ldots \wedge c_m.$$

  The form of a constraint depends on the type of the constrained feature. Constraints on nominal features are equality constraints, those on numerical features are range constraints, those on symbol set features are subset constraints, and those on text features are substring constraints.
  For example, a logical query searching for those restaurants in Trento that are open on Saturday and Sunday is written as: $Q_L=(x_3=\text{"Trento"})\wedge(x_5\supseteq\{7,1\})$.
- The similarity-based query is represented by a probe product, i.e., a vector of feature values, in the same vector space with the product representation.

  $$Q_S = (p_1, \ldots, p_n)$$

  The similarity-based query is matched with the products contained in the catalogue, according to a heterogeneous Euclidean Overlapping distance metric (Ricci et al. 2002). Note that, a feature p_i may be unknown, denoted as "?". In the similarity computation, all of such unknown features will be ignored.
  For example, the similarity-based query Q_S=(?, "pizzeria", ?, 20, ?) means that only two features ('type' and 'cost') will be used in the similarity computation, whereas the remaining ones are unknown.

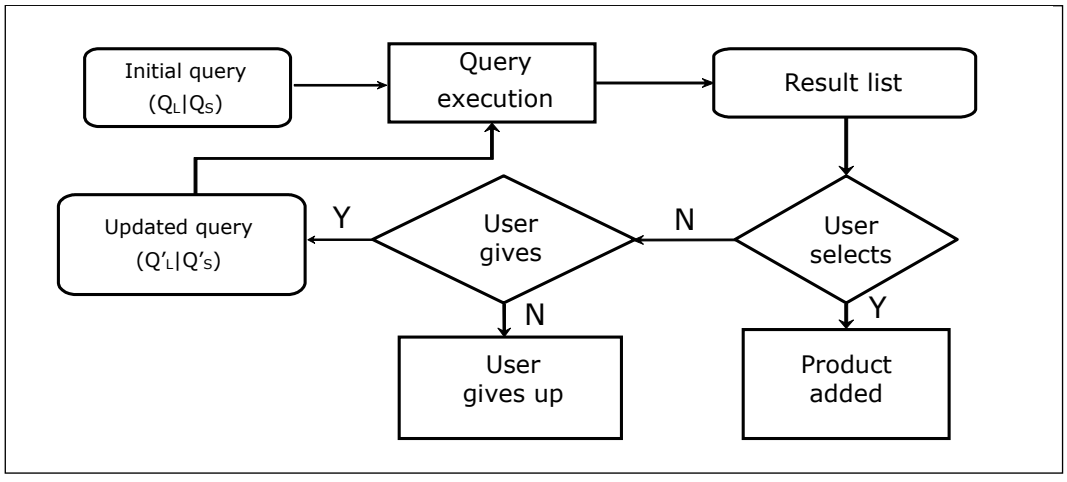

Figure 3: The query updating process

We now sketch how given a query $(Q_L|Q_S)$ and a product $l=(l_1,\ldots,l_n)$ on which the user gives his feedback, the query can be updated to retrieve a new set of products.

is the overview of the query updating process in mITR. The ranked list of products recommended by mITR are browsed by the user; and either a product is selected or feedbacks (critiques) are given on one or more products displayed. The user could give his feedback on a product or on a feature of a product.

Because of the limitations of the mobile environment, especially those related to the screen size and the transmission reliability, it is likely that the number of feedbacks on each iteration (epoch) is limited to a very small value. In particularly, in the mITR system, we limit the number of feedbacks to only one; that is, on each recommendation cycle, the user could give his feedback on only one product or one feature. Moreover, we assume that the feedback is explicitly given by the user and not induced by the system using some implicit indicator (e.g. time spent to look at the product).

In our approach, we assume that a feedback relates to one of two kinds of decision criteria: non-compensatory and compensatory (Edwards and Fasolo, 2001). Non-compensatory criteria represent conditions that should be satisfied completely and independently. Such criteria are encoded (as constraints) in the logical query. On the other hand, compensatory criteria represent conditions that would be satisfied at some degree (not completely), provided that an overall integration function computed on the totality of them is maximized. Compensatory criteria are naturally encoded in the similarity-based query.

We firstly consider the situation when the user gives his (one) feedback on a feature of a product.

F1 - *Positive feedback on nominal feature*: The user states that he likes the value l_i of the i-th feature of a product $l=(l_1,\ldots,l_n)$, and wants to see some more products having a similar value of this feature. Then the new value l_i is assigned for the i-th feature in the similarity query (Q_S).

$$p_i = l_i.$$

For example, if the user says that he likes the restaurant $l=(l_1=$"La Berta", $l_2=$"pizzeria", $l_3=$"Trento", $l_4=15$, $l_5=\{7,1\})$ because of the second feature ($l_2=$"pizzeria"), then the second feature of the similarity-based query will be assigned this value, i.e., $p_2=$"pizzeria".

F2 - *Positive feedback on numerical feature; want less*. The user states that he generally likes a product, but wants to see some more products which should have a smaller value with respect to the i-th feature. Let's call l_i the i-th feature value of the feedback product. Then the old constraint on the i-th feature is removed (if it was present before) and the following constraint is added to the logical query (Q_L).

$$c_i \equiv (x_i \leq l_i - \delta);$$

where δ (>0) is a small adjustment factor which helps to retrieve other products rather than the currently feedback one.
For example, let's assume that the user has provided this type of feedback, saying that he wants something less expensive than 15; and let's assume that the adjustment amount $\delta=1.5$; then the new constraint $c_4 \equiv (x_4 \leq 13.5)$ will be added to Q_L.

F3 - *Positive feedback on numerical feature; accept more*. The user states that he wants to see some more products which would have a greater value with respect to the i-th feature. The old constraint on the i-th feature is removed (if it was present before) and the following constraint is added to the logical query (Q_L).

$$c_i \equiv (l_i + \delta \leq x_i);$$

For example, if the user says that he could accept restaurants even farther from his position this rule can be applied.

F4 - *Positive feedback on symbol set feature*. The user states that he likes the value l_i of the i-th feature of a product, and wants to see some more products having a similar value of this feature. Note that this i-th feature is the type of symbol set. If there was already a constraint on this i-th feature in the logical query it is discarded, and the following constraint is added to Q_L.

$$c_i \equiv (x_i \supseteq l_i)$$

For example, this rule can be applied when the user provides his feedback on the 'openingDays' feature. In this case, he may want to see only those restaurants which are open on some particular days.

F5 - *Negative feedback on nominal feature*. The user states that he dislikes the value l_i of the i-th feature of a product, and wants to see some products not having such value of this feature. Note that this i-th feature is nominal. If there was a constraint on the i-th feature in the logical query, it is removed; and the following constraint is added to Q_L.

$$c_i \equiv (x_i = \text{not } l_i)$$

Next, we consider the situation when the user gives his (one) feedback on a product without specifying its features.

F6 - *Positive feedback on product*. The user states that he wants to see some more products which are similar with a product. Let's assume the judged product is $l=(l_1, ..., l_n)$. Then this product becomes the pattern in the similarity-based query. In other words, each probe feature of Q_S is assigned the value of the corresponding feature of the judged product.

$$p_i = l_i, \text{ for all } i=1,...,n.$$

During the query updating process, if Q_L is updated, then the set of matching products is re-calculated; whereas, if the update is on Q_S, then the set of candidate products is re-ranked (i.e., no change of products, but only a change of their positions in the ranked list). Note that, in any case the user perceives a change in the interface since only five products are presented to his/her judgment.

The objective of the query update is to achieve modified queries which will retrieve products closer and closer to the user needs. In other words, the subsequent query $(Q'_L|Q'_S)$ always reflects the user needs more exactly than the previous one $(Q_L|Q_S)$.

3.2 An example of restaurant recommendation

In this section, we consider an example of restaurant recommendation in the mITR system. The major screens of user interaction are shown in Figure 4.

The user starts the interaction by providing a few of initial constraints. Typically, these constraints are on the 'location' and 'openingDays' features (these contextual information could be derived from other applications running on the device, i.e., a GPS and an agenda). Let's assume that he has not registered yet to the NutKing system. The initial queries (logical and similarity-based), which represent the user's needs, are as follow:

$$Q_L = (x_3 = \text{"VALLARSA"}) \wedge (x_5 \supseteq \{2\})$$

and

$$Q_S = (?, ?, \ldots, ?)$$

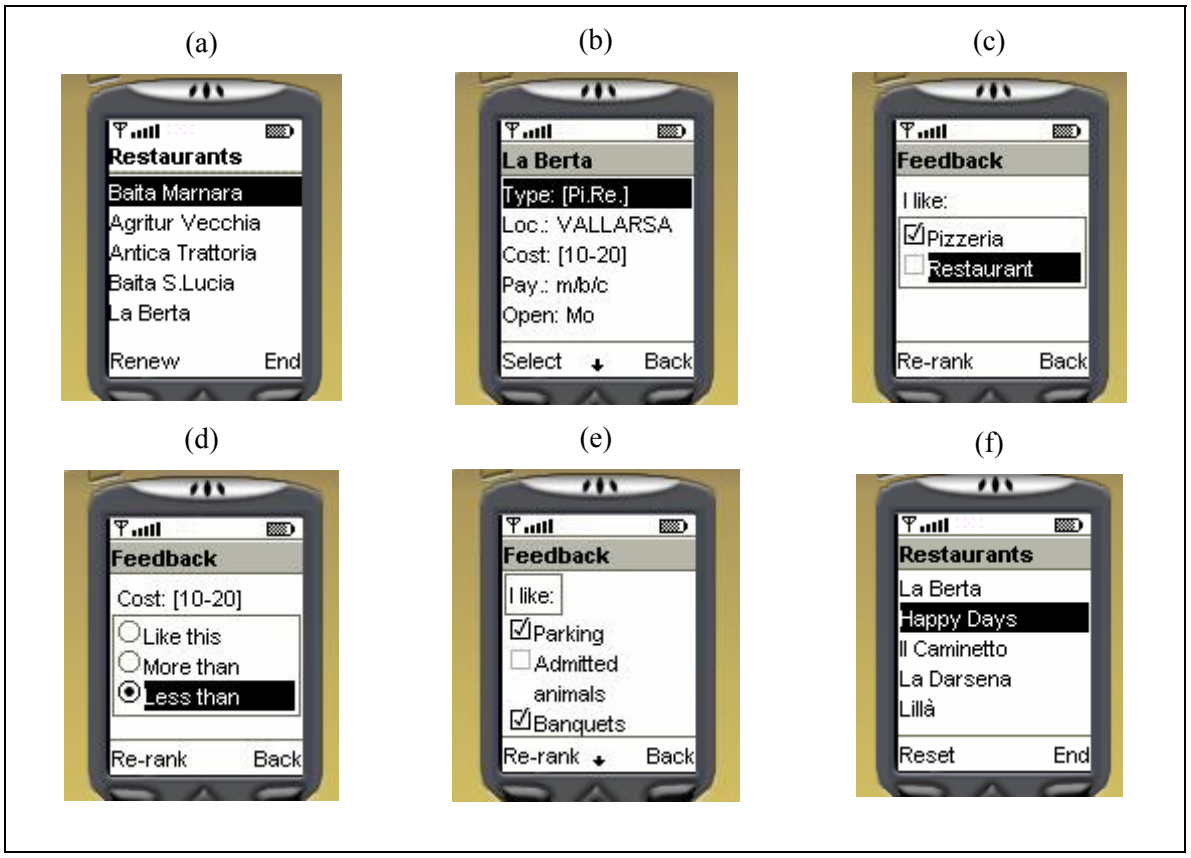

Figure 4: restaurant recommendation

These two initial queries (Q_L and Q_S) are processed in order to produce an initial set of candidate restaurants. These candidate restaurants then are present to the user, as in Figure 4a. Because of the limitation of screen of mobile devices, only top 5 restaurants (i.e., those satisfy Q_L and have the highest similarities with Q_S) are displayed. In the case there are two restaurants having the same value of similarity with Q_S, the one having lower cost appears first. Let us consider three matching restaurants:

- (x_1="Baita Marnara", x_2="trattoria", x_3="VALLARSA", x_4=5, x_5={1,2,3,4,5,6,7})
- (x_1="La Berta", x_2="pizzeria", x_3="VALLARSA", x_4=10, x_5={2})
- (x_1="Happy Days", x_2="pizzeria", x_3="VALLARSA", x_4=15, x_5={2})

In the result of the first recommendation cycle (the initial one), the "Baita Marnara" restaurant appears at the first position, the "La Berta" restaurant at the fifth position, and the "Happy Days" restaurant is out of the top-5 list. For those restaurants that satisfy the query Q_L, the more similar they are to the query Q_S, the higher position they appear in the ranked result list (see Figure 5). Note that in the case two restaurants having the same similarity value, the cheaper will appear above the remainder.

The mITR system provides an interface which allows the user to view a restaurant's detailed description (Figure 4b). On the screen showing a restaurant's detailed description, those features which are wrapped by square brackets are feedback-enabled; that is, the user could express his judgments on such features. Three example situations are shown in Figure 4c, d, e. In Figure 4c and Figure 4e, the user provides an F1 feedback. In Figure 4d, the user gives his feedback on the

cost of an interested restaurant. Here the two types of feedbacks “More than” and “Less than” (F3 and F2) are illustrated.

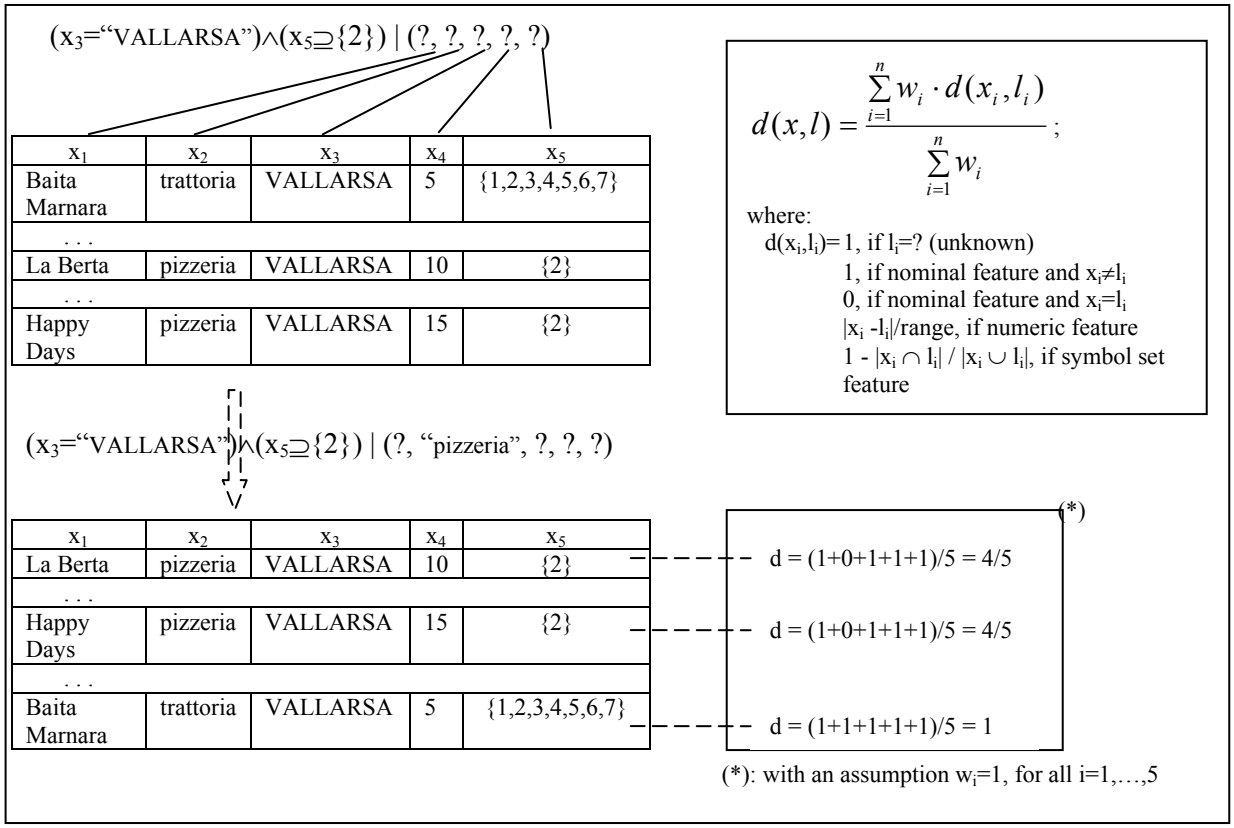

Figure 5: query update example

Suppose now the user says that he wants to see restaurants similar to the restaurant “La Berta”, and provides his feedback on the feature ‘type’ (Figure 1Figure 4c). This feedback (type F1) is then incorporated in the similarity-based query. The updated query is then processed in order to produce a new set of candidate restaurants, as shown in Figure 4f. Note that, in the new recommendation, the “Happy Days” restaurant is pushed up to the second position; whereas the “Baita Marnara” restaurant goes out of the top-5 list. Figure 5 depicts how the new (updated) query is processed in order to produce the new list of recommended restaurants.

Such recommendation cycles go on until the user could find out his favourite restaurant; or he gives up after a recommendation cycle.

3.3 Technology used and why

The proposed system has been implemented on a portable device. The integration between the mobile component and the server site is achieved by using the J2ME technology, on the client side, and a J2EE application server with a relational database on the server side. The system uses XML for data exchange purposes.

The Java technology can run on devices which are disconnected from the network. Because the Java technology allows for disconnected operations, the developer can choose which parts of the business logic are executed on the client (i.e., a mobile device) or on the server. This allows developers to create graphical applications, including office applications, and more. The Java

technology also provides a robust, well-tested security model, with many J2ME-based devices supporting end-to-end secure HTTP (HTTPS).

J2ME also has a generic connection framework, which allows to connect your phone to other devices, like a digital photo camera or a GPS device. If your device for example has a serial port, or is Bluetooth enabled, J2ME applications can easily access the data from the photo camera and store them locally in a database and synchronize the database with a server or use GPS data to obtain the position.

The Java language is a fully developed and well-understood programming language, and executes safely and portably on various mobile and wireless devices. This is a critical strength for the Java platform, so developers can write J2ME applications once with the certainty that it can be deployed and run correctly on other compliant devices available to consumers.

For consumers, Java-enabled interactive services are the next step beyond today's text-based static content. Java software enhances the user experience by supporting easy-to-use, graphical, interactive services for wireless devices. Examples include:

- Dynamically generated, personalized stock quotes that can display graphs and give purchase and/or sell alerts for specific stocks utilizing the wireless network efficiently.
- Real-time, location-specific weather reports that display periodic forecast updates.
- Real time, location specific traffic reports that update local traffic conditions and supply alternate highway routes depending on traffic delays and accidents.
- Games that can be downloaded and played offline by individual users achieving cost effective use of the wireless network.

4 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an integrated recommender system that provides support for pre-travel planning and on-tour travel plan update. We have described the approach used in the mITR system that focuses on on-tour travellers. The mITR system extends and co-operates with the NutKing system (Ricci et al. 2002), aimed at supporting a traveller in information filtering and product bundling.

We are now conducting a usability evaluation of the approach used in the mITR system. We must understand if the proposed approach of recommendation based on users' interactions and feedbacks is appropriate or not. We decided to start the evaluation on some few popular kinds of devices; e.g., Java-enabled mobile phones and Java-enabled pocket PCs. This evaluation, obviously, will consider the user interface, and is aimed at understanding which kind of information should be shown to the users, in which format and in what moment.

Moreover, in case the user is providing a feedback on a product (the feedback type F6), our current solution is to incorporate all features of this product into the new similarity-based query. Obviously, this decision could be improved; since many of these features could not reflect the user's interest, and could cause a 'misleading' update. To improve this coarse update rule, it is necessary to determine which (important) features of the feedback product(s) should be incorporated in the new query.

Before being presented to the user, the products suggested by mITR are ranked in an order of the fitness to the user's needs. This ranking relates to the computation of the similarity between a candidate (recommended) product and the similarity-based query. Since in mITR a product is represented by its features, the similarity evaluation is a weighted sum of local similarities between each pair of values corresponding to a feature. In fact, different features have different

important weights in a certain session (or in a long period of time). At the moment, mITR considers all features having the same importance. In the future, we expect to improve this simple assumption.

Another important aspect of feedback-based updating of the user query is the precise modality offered to the user to enter his feedback; i.e., explicit vs. implicit. They require different implementations, and produce very different results. At the moment, the mITR system is exploiting only those user preferences that have been explicitly acquired. Explicit preferences have the advantage of representing accurately users' interests. However, the disadvantage is the cost (from the view point of users) in providing such explicit preferences. Mobile users do not like to be asked or to provide much information. This is the reason why mITR allows only one feedback on each iteration step.

Conversely, implicit feedbacks require nearly no cost on the user effort since they usually are extracted from the information of users' interaction history. In the future development, we plan to find out an appropriate method to exploit such implicit preferences in initializing the query.

Another issue is related to negative feedbacks. In the real environment, users do not always give positive feedbacks (e.g., "I like it" or "I like this feature, but more..."). In some cases, he may want to say something like "I dislike it" or "I dislike this feature".

As final remark, we must notice that the text-typed features (e.g., the 'description' feature) have not been incorporated into the similarity-based query. To do that, we need to find some appropriate metric to compute the similarity between two texts. The results in the Information Retrieval field would be useful in order to reach this objective.

REFERENCES

Burke, R. (2000). *Knowledge-based recommender systems*. In J.E. Daily, A.Kent, and H.Lancour, editors. Encyclopedia of Library and Information Science, volume 69. Marcel Dekker.

Edwards, W. and Fasolo, B. (2001). *Decision Technology*. Annual Review of Psychology, 52, p.581-606.

Fesenmaier, D. and Jeng, J. (2000). *Assessing structure in the pleasure trip planning process*. Tourism Analysis, 5, p.13-17.

Ginty L. M. and B. Smyth (2002). *Deep dialogue vs casual conversation in recommender systems*. In Recommendation and Personalization in eCommerce, Proceedings of the AH'2002 Workshop, p.80-89, Malaga, Spain. Universidad de Malaga.

Hwang, Y.-H., Gretzel, U. and Fesenmaier, D.R. (2002). *Behavioural Foundations for Human-Centric Travel Decision-Aid Systems*. In: Wöber, K.W., Frew, A.J. and Hitz, M. (eds.) Information and Communication Technologies in Tourism. Springer: Wien, p.356-365.

Ricci, F., Arslan, B., Mirzadeh, N. & Venturini, A. (2002). *ITR: a case-based travel advisory system*. In S. Craw and A. Preece (eds), 6th European Conference on Case Based Reasoning, ECCBR 2002, Springer Verlag, p.613–627.

Shimazu, H. (2001). *Expertclerk: Navigating shoppers buying process with the combination of asking and proposing*. In B. N. (Ed.), editor, Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, p.1443-1448. Morgan Kaufmann.

Werthner, H. and Klein, S. (1999). *Information Technology and Tourism - A Challenging Relationship*. Springer Verlag.