

Context-Aware Personalised Service Delivery

E Pignotti and P Edwards and G A Grimnes¹

Abstract. In this paper we explore the potential of recommendation systems in an environment where users can access a variety of services from different locations. In particular we explore the use of adaptive technologies to recommend new services to users based on context, which we define in terms of four factors: time, location, user behaviour and user profile. A multi-agent service delivery architecture is used as a platform for the recommendation system. User behaviour in the service environment is analysed and used to predict the next service that a user will invoke; the user's preferences and past behaviour are then utilised to recommend specific aspects of that service to the user.

1 Introduction

Organisations have traditionally enabled users to locate Web services through search engines and portals; the recent explosion of such resources makes discovery of relevant services difficult. Recommendation techniques [7] can be used to help people locate resources of potential interest; one mature application area being TV programme recommendations, e.g. PTV [2].

In addition to the service/information overload problem, the recent growth of wireless communication technologies allows increasing numbers of users to access services anywhere and at any time. This requires new factors such as location and time to be considered in the process of recommending available resources. In this paper we discuss a system which employs a number of recommendation strategies in an environment where users can access a range of services. Central to our approach is the use of adaptive technologies to recommend services to users. Such recommendations can only be made with a full understanding of context; we consider four factors:

- Time: Date and time when the user accesses a service.
- Location: The user's location (home, office, ...).
- Visited Services: History of services visited by the user.
- User Profile: Service specific user preferences.

In this paper we also explore the use of existing user profiles to further personalize [6] the content of a recommendation message. Use of RDF² as the service content language facilitates interoperability with another system which provides these profiles.

We are taking the idea of delivering resources in a dynamic environment a step forward by using recommendation techniques to facilitate service prediction and utilisation. We have implemented a system called RECO which aims to facilitate the use of context-aware services in a dynamic environment. RECO integrates with popular Web browsers through a client application; particular effort has been

devoted to implementing a client for Microsoft Pocket PC which demonstrates the potential of the system where users can freely move from location to location; see Figure 1.

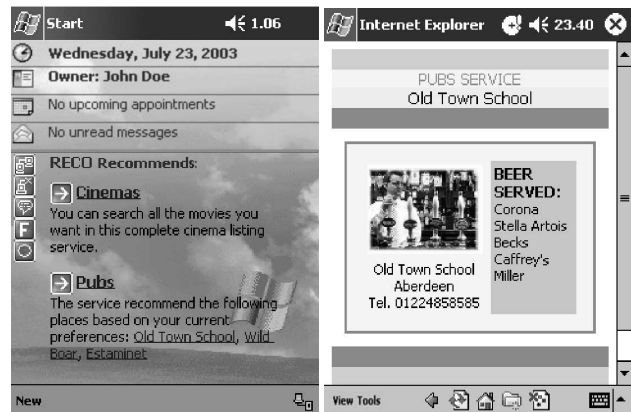


Figure 1. The RECO Client Application for Microsoft Pocket PC.

As a context for our discussion, consider the following use case: an operating environment provides a range of information services giving details about pubs, cinemas and seminars. Users are able to access these services from different locations and at different times. Patterns can be identified in user behaviour, e.g. when an individual searches for a restaurant he also searches for a movie for the same evening; if he is in his office and uses the seminar service he may also look for a pub. Such a model could be used to assist the user by basing recommendations on the services already visited.

1.1 Managing Context

Context clearly depends upon the location from which a user is accessing the system, the current time, their user profile and patterns of service use. The choice of profile mechanism to use in RECO was influenced by our previous work on an agent-based multi-service environment - GraniteNights [3]. GraniteNights defines a portable and re-usable user preference profile encoded in RDF, with appropriate ontology support in order to facilitate the re-use of the profile in other applications. These profiles are generated from previous user queries to the system and from any feedback they might have given on the result of those queries [3].

User behaviour within RECO is recorded into a system log (see Figure 2); in order to use this information for recommendation purposes it is necessary to extract patterns of service use. Two learning technologies are used to identify patterns in user service invocation,

¹ Department of Computing Science, Kings College, University of Aberdeen, Aberdeen, AB24 3UE, Scotland. Email: {epignott, pedwards, ggrimnes}@csd.abdn.ac.uk.

² <http://www.w3.org/RDF/>

and patterns in user location/time. Sequence analysis [12] identifies sequential patterns in a large volume of transaction data; market basket analysis being a typical application. A sequential pattern consists of a sequence of items that frequently occur in the training data provided. In RECO a possible pattern could be: RESTAURANTS → CINEMAS. This indicates that when the user visited the Restaurant service he also subsequently visited the Cinema service. In order to discover sequential patterns in user behaviour, the *AprioriAll* algorithm was selected [12]. Space does not permit us to discuss the algorithm in detail. Proximity rules are used to define if the user is in the right location and at the right time to receive a recommendation from the system. The NNGE [8] algorithm was selected for this role as it gave the best results when evaluated against a set of artificial training data [5]. Example proximity rules generated by the system are shown in Figure 7.

In a community of users it is possible to identify two kinds of service invocation pattern: one represents the behaviour of a single user in the system; the other represents the behaviour of the entire user community. Knowledge about the community of users can be used to generate recommendations for a user new to the system. Ultimately, context is managed within RECO by monitoring user behaviour, i.e. what services are invoked, in what sequence, where from and at what time.

| UserID | Session | Sequence | ServiceID | Location | Time |
|--------|---------|----------|-------------|----------|------|
| John | 1 | 1 | Seminars | Office | Mon |
| John | 1 | 2 | Pubs | Office | Mon |
| John | 2 | 1 | Seminars | Office | Thu |
| John | 2 | 2 | Pubs | Office | Thu |
| John | 3 | 1 | Restaurants | Office | Fri |
| John | 3 | 2 | Cinemas | Office | Fri |
| John | 3 | 3 | Pubs | Office | Fri |
| John | 4 | 1 | Restaurants | Home | Sun |
| John | 4 | 2 | Cinemas | Home | Sun |
| John | 4 | 3 | Pubs | Home | Sun |

Annotations: A bracket on the right side groups the first three rows as 'TRANSACTION' and the next three rows as 'SEQUENCE'. An arrow points from the 'SEQUENCE' label to the 'Sequence' column. A bracket at the bottom groups the last three rows as 'ITEMSET' and the 'ServiceID' column as 'ITEMS'.

Figure 2. Example Log Data.

2 RECO Architecture Overview

RECO is designed to recommend context-aware services to the user based on an existing user model. The model comprises the sequence rules and proximity rules generated by RECO, plus an existing RDF user profile. In this section we explore the design of various components, including the multi-agent architecture, and the recommendation techniques integrated within the system. The first recommendation technique analyses the user’s behaviour in the environment and predicts the next service that (s)he will invoke; the second utilises a profile of the user’s preferences to recommend more specific information.

2.1 The Multi-Agent System

At the core of the RECO system is a multi-agent architecture which provides the agents that are needed in order to monitor user behaviour and make recommendations. In order to communicate all RECO agents use standard FIPA³ compliant messages envelopes,

³ FIPA is a non-profit organisation aimed at producing standards for the interoperation of heterogeneous software agents (www.fipa.org).

with content expressed in XML. Figure 3 presents the overall system architecture; the main agent roles will now be described:

- **Environment Agent** : Responsible for keeping a list of users and services currently connected to the system, it also monitors changes in the environment when new users register or users access a service. The Environment Agent holds a “log” of all the services that a user visits (see Figure 2), and the AprioriAll and NNGE algorithms are continuously applied to the log in order to identify a model of user behaviour. This model is used to predict if a user will access a service in the near future, given the current time and location; such a prediction is used to alert the appropriate service agent which will then attempt to recommend a personalised version of the service content to the user.
- **Service Agent** : Each service agent is an extended version of the standard information agent platform developed by the GraniteNights project [3]. Each agent implements a query interface based on RDF Query By Example (QbEX) [3]. When the Environment Agent predicts that a user will use a service in the near future it alerts the relevant service agent. Once the service agent receives such an alert it begins by initiating a dialogue with the appropriate user agent to retrieve the user’s profile. The response from the user agent reflects the user’s preferences for the relevant service and consists of a list of previous queries performed by the user. Figure 4 contains an example of one such past query (highlighted). The service agent next compares all the past queries against the information resources currently managed by the service. The results returned by the different queries are then combined and sorted by the number of examples (previous queries) supporting each result. QbEX plays an important role in extracting such results; in fact, the historical queries are simply used to re-query the service. To create RECO service agents, the following features were added to the GraniteNights information agent: accept alert messages from the Environment Agent; request user profiles from the relevant user agent; send recommendation messages back to the user agent. The recommendation message contains the service reference, a text description and a list of links. An example message is shown in Figure 6.
- **User Agent** : The user agent manages interactions with the user through the client application; in addition, it handles the user profile, an example of which is shown in Figure 4. The queries contained in the profile represent previous examples of user behaviour and include appropriate ontology references. When contacted by a service agent, the user agent extracts only

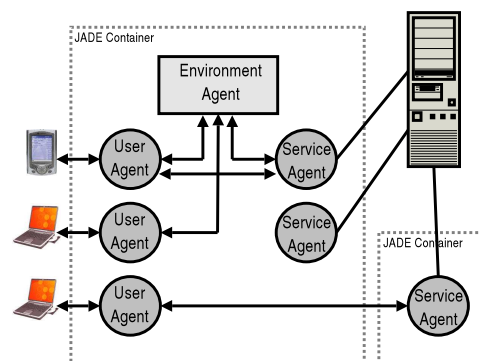


Figure 3. Multi-Agent System Architecture.

```

<rdf:li>
<j.2:Interaction
  <j.2:timestamp="20030808T153338">
  <j.2:pref>
  <rdf:Seq>
  <rdf:li>
  <rdf:li>
  [
    <j.4:Query>
    <j.4:template>
    <j.0:Restaurant>
    <j.0:typeOfCuisine rdf:type=
      "restaurant-v4#ChineseCuisine"/>
    </j.0:Restaurant>
    </j.4:template>
  ]
  </j.4:Query>
</rdf:li>
<rdf:li>
<j.4:Query>
<j.4:template>
<j.5:EnglishPub>
<j.5:servesBeer rdf:resource=
  "beertypes#corona"/>
<j.5:liveEntertainment rdf:resource=
  "pubs#Never"/>
</j.5:EnglishPub>
</j.4:template>
</j.4:Query>
...

```

Figure 4. Example User Profile.

those parts of the profile that match the appropriate service ontology and transmits those. The user agent is also connected to the user's input device and informs the Environment Agent of the user's actions. The user agent is also responsible for dispatching messages back to the user device once it receives a recommendation from a service agent.

2.2 Service Predictor & Alert System

The Service Predictor and Alert System is a module within the Environment Agent. Its task is to predict the next service that a user will invoke; it also must check if the time and user's location are appropriate for that recommendation. Figure 5 shows the architecture of this module.

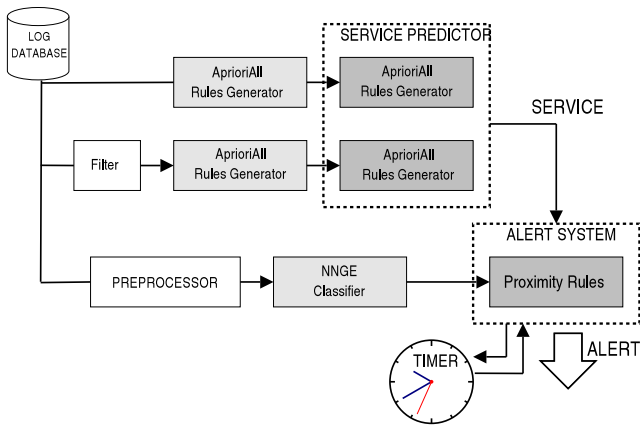


Figure 5. Service Predictor and Alert System Modules of the Environment Agent.

As explained above, details of user behaviour in the environment are logged; this information is processed by the AprioriAll algorithm

in order to create a set of sequence rules, which reflect patterns in service invocation (Table 1). Each rule has an associated support and a confidence level dependent upon the number of examples in the log data supporting that rule. These rules are generic for the entire environment because they are generated from the log data of all users in the system. The same sequence analysis procedure is applied to each user of the system in order to create more specific (user-centred) service sequence rules.

The two sets of sequence rules (environment and user) are then used by the Environment Agent to predict the next service that a user will access; the context for this prediction is provided by the services already visited by the user during the current session. The system identifies rules that can be applied (by matching the lefthand side) and uses the one with the highest support and confidence values to suggest the service that the user may next access. When the Environment Agent extracts the applicable rules it gives priority to the user sequence rules. In other words, if a user rule matches the current user session, the environment service sequence rules are ignored. However, environment sequence rules can be used if a user has never explored certain services or if the user is using the system for the first time.

As explained above, the system's aim is to recommend services to the user at the right time and location. Sequence rules alone are insufficient, because time and location are not considered when they are generated. In order to overcome this problem a set of proximity rules are also generated from the log data using the NNGE algorithm. Figure 7 presents some examples of proximity rules generated by RECO. If the Service Predictor predicts the Restaurant service as the next service, the second rule in Figure 7 is used against the session information; if the user is at home, the day is Friday, Saturday or Sunday and the time is 17:25 (represented as 1045 minutes) the proximity rule supports the service prediction. In this case, an alert message is then sent to the appropriate service. If the proximity rule is not valid for the current situation the Environment Agent continues to monitor the user until time and location are correct; during this period the Service Predictor may of course identify a different service.

2.3 Client Application Design

The design of the client application is not an issue in this system as different client applications can be developed for different platforms and devices. However, the client applications have to respect certain guidelines and a specific communication protocol. The implementation language for the client application must be able to handle socket connections and parse XML messages. The client application must also be able to display recommendations received from the system in HTML. Figure 1 shows an example of a client application for Microsoft Pocket PC, illustrating two different styles of RECO recommendation. The first of these suggests the Cinemas service using a standard message; while the second suggests the Pubs service with some specific personal suggestions. The system has utilised user preference information for the Pubs service to generate these recommendations. Figure 1 also show an example where the user has followed a specific recommendation.

2.4 System Implementation

RECO was implemented using the agent platform JADE [1]. JADE was selected because RECO required a FIPA compliant platform to build upon existing services developed for the GraniteNights project.

```

<RECOMMEND>
<SERVICE_ID>1001</SERVICE_ID>
<SERVICE_NAME>pubs</SERVICE_NAME>
<DESCRIPTION>
<![CDATA[The service recommends the
following places based on your
current preferences:
<A href = "10010">Old Town School (23)</A>,
<A href = "10011">Wild Boar (20)</A>,
<A href = "10012">Estaminet (17)</A>]]>
</DESCRIPTION>
<URL>
http://localhost:8080/reco/pubs.jsp
</URL>
<LINKS>
<ID>10010</ID>
<LINK>
http://localhost:8080/reco/pubs.jsp
#wild_Boar
</LINK>
</LINKS>

```

Figure 6. Example of Recommendation Message.

The core of the RECO system was developed using Java 2 with Delphi and embedded Visual C++ used to create client applications for Internet Explorer and Microsoft Pocket PC.

3 Illustrative Example

In order to illustrate the potential of the system we now present a usage scenario. To recommend services efficiently RECO needs training information which requires the system to be utilised for several weeks by several users. In the example presented here we have therefore assumed that log data has been analyzed to generate the initial rule sets shown in Table 1 and Figure 7. These rules form the basis for an initial environment service model.

The analysis of the behaviour of the system can be considered from four perspectives:

- How the system reacts with a new user in the environment;
- How the system behaves when an existing user uses the system on repeat visits;
- How the system behaves when there are pending recommendations for a user;
- How the system reacts if an existing user discovers a new service.

```

1 class cinemas IF : location in {home}
  ^dayOfWeek in {fri,sat,sun}
  ^time=1055.0 (15)
2 class restaurants IF : location in {home}
  ^dayOfWeek in {fri,sat,sun}
  ^time=1045.0 (15)
3 class pubs IF : location in {home,office}
  ^dayOfWeek in {mon,tue,wed,thu,fri,sat,sun}
  ^945.0<=time<=1035.0 (36)
4 class seminars IF : location in {office}
  ^dayOfWeek in {mon,tue,wed,thu,fri}
  ^time=510.0 (21)

```

Figure 7. Proximity Rules.

When a new user begins to use the system, there are no user-specific sequence rules available; as a result, the environment model is used. Imagine that a user (id:epignott) accesses the Restaurants

| Initial Rules | Sup. | Conf. |
|----------------------------------|------|-------|
| [pubs, restaurants] -> [cinemas] | 41% | 100% |
| [pubs]->[cinemas] | 41% | 41% |
| [restaurants]->[cinemas] | 41% | 100% |
| [seminars]->[pubs] | 58% | 100% |
| Final Rules | Sup. | Conf. |
| [pubs, restaurants] -> [cinemas] | 32% | 100% |
| [pubs]->[cinemas] | 32% | 39% |
| [restaurants]->[pubs] | 32% | 60% |
| [restaurants]->[cinemas] | 32% | 60% |
| [restaurants]->[evenings] | 15% | 28.0% |
| [seminars]->[pubs] | 45% | 95% |

Table 1. Environment Rules: Initial & Final Rule Sets.

service from home. Based on the environment service rules the system selects the Cinema service. Moreover the system checks if the service is appropriate for the user's current location and time using the proximity rules. Rule 1 (Figure 7) fires confirming the recommendation (as it is Friday at 17:35).

The user ignores this recommendation, and instead looks for a public house using the Pubs service. Following these interactions the system is able to build a specific model for this user as shown in Table 2 (New User). In addition, this user's service interactions have caused the environment model to change; the new rule: [restaurants] -> [pubs] is added to the environment model and the support and confidence of the other rules involving restaurants and pubs decreased accordingly.

In the second case the user *epignott* returns to the system, and the Environment Agent retrieves the user model acquired earlier (Table 2). Imagine that the user once again accesses the Restaurants service. The system now recommend the Pubs service based on the user model and the proximity rules. In this case, the environment model is ignored.

In the previous visit to the Pubs service the user searched for a pub serving Corona beer; the user profile thus contains a representation of that past query (see Figure 4). The Service Agent retrieves the past query to search in the Pubs resource for places serving Corona. The results of the search are integrated into the recommendation (Figure 6). Figure 1 presents those recommendations to the user.

If the user considers the recommendation and decides to visit the Pubs service the system automatically updates the user and environment models. No new rules are added to the user model but the support of the existing rule is increased, as shown in Table 2. No new rules are added into the environment model, but the support and confidence of similar rules are adapted to reflect the new behaviour.

In the third case imagine that the user once again accesses the Restaurant service. Based on the user model the system would normally recommend the Pubs service. However, on this occasion the location of the user and the current time do not match any proximity rule, and the system will not recommend the Pubs service. The recommendation for this service does not appear, but remains 'pending' until the user's context satisfies one of the proximity rules.

In the last case the user *epignott* visits again after a while, and discovers that a new service is available. This is the Evening service [3] which is able to plan an entire evening for the user. Initially he visits the Restaurants service to select a dining location. The system suggests looking at the Pubs service based on the user model generated in the past. Instead, the user decides to visit the new Evening service to plan his evening. This causes a new rule to be added to the user model (Table 2).

The next time the user follows the same path, the system will still recommend the Pubs service because the specific rule still has higher

support. However, after several episodes in which the user follows the path to the Evening service, the system learns the change of behaviour and starts to recommend that service. Table 1 shows that the new rule ([restaurants]->[evenings]) is also added into the environment model; however the support and confidence of the rule are initially not high enough to be considered for a new user.

| <i>New User</i> | Sup. | Conf. |
|----------------------------------|------|--------|
| [restaurants]->[pubs] | 2% | 100.0% |
| <i>Returning User</i> | Sup. | Conf. |
| [restaurants]->[pubs] | 5% | 100.0% |
| <i>Discovering a new service</i> | Sup. | Conf. |
| [restaurants]->[pubs] | 3% | 70.0% |
| [restaurants]->[evenings] | 1% | 20.0% |

Table 2. Service Rules for User epignott: As a New User, Returning User and Discovering a New Service.

4 Related Work

This paper has presented an intelligent multi-agent system for context-aware service delivery and recommendations to mobile users. A system with very similar goals is ACCESS [4] developed at University College, Dublin. In ACCESS, context is considered an aggregation of the user's location, their previous activities and their preferences. The user preferences are acquired through a Web-form completed manually by the user when (s)he joins the system. RECO extends the ACCESS concept of context by also attempting to situate each atomic service task within a sequence. In addition, RECO will also automatically learn a user's preferences over time, avoiding the necessity of filling in forms when starting to use the system. By using environment rules RECO is still able to provide service recommendations even when a user has just joined the system.

Another similar system is ClixSmart Navigator [11] which aims to automatically adapt the structure of a portal to facilitate access from a mobile device. ClixSmart implements a Navigation Server between a content store and a WAP gateway. In RECO we replace the standard browser with an agent-based client which allows the dispatch of recommendation messages without an explicit request from the user. Moreover the client is integrated into the device and its use is totally transparent to the user (see Figure 1).

Another central feature of RECO which is lacking in both of these systems is the use of standard ontologies, protocols and communication languages. By choosing RDF as the service content language we have already made interoperability with the GraniteNights system possible. In addition, by also using the standard ontologies from the Agentcities⁴ project, RECO may interoperate with other FIPA based agent platforms.

At the heart of RECO's recommendation module is the Apriori-All algorithm used to identify common service invocation sequences. The standard application of AprioriAll is market basket analysis, to identify products with strong causal links between them (such as diapers and beer). We believe that RECO represents the first use of AprioriAll to detect sequential patterns of service invocations, and we feel there is much to gain by such an analysis, from predictive network caching to one-click user recommendations. Similar ideas have been used in wireless networks to predict that a user will access a resource from a particular cell, allowing the network to transfer the resource to the cell before the user accesses it [10].

⁴ <http://www.agentcities.org>

5 Discussion & Future Work

The work described in this paper has successfully employed adaptive techniques to recommend services to a user based on use of service behaviour and known preferences. The use of sequence analysis techniques enables the system to predict which service a user will access in the near future, while proximity rules allow the system to deliver personalised information if the user accesses a service at the right location and time. By capturing two different models of user behaviour (environment sequence rules and user sequence rules) the system can deliver service recommendations even if a user has never accessed the system. Our work to date leaves room for future expansion. At the moment if a particular user has never visited a service, the system can still recommend that service to the user based on environment sequence rules. However, the recommendation from the service is not personalised, as there are no previous queries in the user profile. One strategy would be to compare a new user's developing sequence rule model with those of other users who also have more developed user preference profiles. Using techniques similar to those used in collaborative recommendation systems [9], aspects of those preference profiles would be shared with the new user.

REFERENCES

- [1] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. Jade : A white paper, 2003.
- [2] P. Cotter and B. Smyth, 'Ptv: Intelligent personalised tv guides', in *Proceedings of the 12th Innovative Applications of Artificial Intelligence (IAAI-2000) Conference*. AAAI Press, (2000).
- [3] G. AA. Grimnes, S. Chalmers, P. Edwards, and A. Preece, 'Granitenights - a multi-agent visit scheduler utilising semantic web technology', in *Seventh International Workshop on Cooperative Information Agents*, (2003).
- [4] C. Muldoon, G.M.P. OHare, D. Phelan, R. Strahan, and R.W. Collier, 'Access: An agent architecture for ubiquitous service delivery', in *Proceedings Seventh International Workshop on Cooperative Information Agents (CIA)*, (2003).
- [5] E. Pignotti, *Intelligent Adaptive Service Delivery*, MSc dissertation, University of Aberdeen, 2003.
- [6] D. Reicken, 'Special issue on personalization', *Communications of the ACM*, **43**, (2000).
- [7] P. Resnick and H.R. Varian, 'Special issue on recommender systems', *Communications of the ACM*, **40**, (1997).
- [8] S. Salzberg, *A nearest hyperrectangle learning method.*, volume 6, 277-309, 1991.
- [9] U. Shardanand and P. Maes, 'Social information filtering: Algorithms for automating 'word of mouth'', in *Proceedings of the ACM Conference on Human Factors in Computing Science*, (1995).
- [10] Xuemin Shen, John W. Mark, and Jun Ye, 'User mobility profile prediction: An adaptive fuzzy inference approach', Technical report, University of Waterloo, (2000).
- [11] Barry Smyth and Paul Cotter, 'Personalized adaptive navigation for mobile portals', *ECAI 2002*, (2002).
- [12] Ramakrishnan Srikant and Rekish Agrawal, 'Mining sequential patterns: Generalizations and performance improvements', Technical report, IBM Almaden Research Centre, (1999).