Spatial Hoarding: A Hoarding Strategy for Location-Dependent Systems

Karim Zerioh[2], Omar El Beqqali[2] and Robert Laurini[1]

[1] LIRIS Laboratory, INSA-Lyon – Bât B. Pascal – 7 av. Capelle F – 69621 Villeurbanne Cedex France Robert.Laurini@insa-lyon.fr
[2] Dhar Mehraz Faculty of Science, Mathematics and Computer Science Department, B.P. 1897 Fes-Atlas, 30000, Fes, Morocco. omar.el-beqqali@liris.cnrs.fr, kzerioh@yahoo.com

Abstract

In a context-aware environment, the system must be able to refresh the answers to all pending queries in reaction to perpetual changes in the user's context. This added to the fact that mobile systems suffer from problems like scarce bandwidth, low quality communication and frequent disconnections, leads to high delays before giving up to date answers to the user. A solution to reduce latency is to use hoarding techniques. We propose a hoarding policy particularly adapted for location-dependent information systems managing a huge amount of multimedia information and where no assumptions can be made about the future user's location. We use the user's position as a criterion for both hoarding and cache invalidation.
Keywords: Hoarding, cache invalidation, mobile queries, location-dependent systems, spatial information systems.

1 Introduction

The growing popularity of mobile computing has lead to more and more elaborated mobile information systems. Nowadays, mobile applications are aware of the user's context (time, location, weather, temperature, surrounding noise, ...). One of the most popular context-aware applications is the tourist guide (Cheverst et al. 2000, Abowd 1997, Malaka 1999, Poslad et al. 2001, Zarikas et al. 2001). In this paper we deal only with location.

Let's consider the scenario of a tourist with a mobile tourist guide asking where the nearest restaurant is located. This query must be answered immediately. Otherwise, if the answer takes some delay, it may be obsolete if the tourist in movement is already nearer to a different restaurant. So the system must refresh the responses that have been invalidated by context changes, to all pending queries.
This operation can be repeated several times, depending on the number of users and the frequency of their queries and the context changes. On the other hand, mobile systems still suffer from scarce bandwidth, low quality communication and frequent network disconnections. All these factors lead to high delays before satisfying user's queries. But this delay will not occur if the answer is already in the client's cache. Caching techniques have proven their usefulness in wired systems. The answer of a query is stored in the cache for future use and when a user asks the same query it is answered by the cache. However, in location-dependent systems, where the answer of the same query changes if only the user's position is different, and where users rarely return to the same place (for example, a user with a tourist guide, after visiting a museum, has a very low chance to return to it after a while), the benefits of caching are not so obvious. But, if useful information is transferred to the client before the user requests it, the problem of latency will be resolved.

Hoarding techniques must predict in advance which information the user will request, and try to transfer as less as possible unusable data for not wasting the scarce bandwidth resources and the usually limited memory and storage capacity in the user's device. Tourist guides, nowadays, are very elaborated. They use maps for guided tours, present audio content to allow the user to walk while listening to explications, provide virtual 3D reconstructions of historical sites and 3D representations of the place where the target asked by the user is being for making its recognition easier, offer a life show of a tour in a hotel… So the amount of the multimedia data dealt with is really huge. This makes the necessity of appropriate cache invalidation schemes for freeing space in the user's device.

In this paper, we present a hoarding technique particularly adapted for location-dependent systems managing an important amount of data (called spatial hoarding). We use the user's location both as a prediction criterion and as a cache invalidation one.

The rest of the paper is organised as follows. In section 2 we begin with the related work. In section 3 we give a description of our method, we define the client's capability and propose to operate in disconnected mode to save power. In section 4, we present two necessary algorithms for implementing the SH strategy, and discuss how data must be organised for the

determination of the information that must be hoarded. Section 5 gives an overview of our future work. Finally, section 6 concludes the paper.

2 Related Work

Caching is the operation of storing information in the user's device after it has been sent from the server. This allows future accesses to this information to be satisfied by the client. Invalidation schemes are used for maintaining consistency between the client cache and the server. Invalid data in the cache may be dropped for freeing memory for more accurate data. In location-dependent systems a cached data value becomes invalid when this data is updated in the server (temporal-dependent invalidation), or when the user moves to a new location (location-dependent). The team of the University of Science and Technology at Hong Kong (Xu et al. 1999, Zheng et al. 2002) investigated the integration of temporal-dependent and location-dependent updates. They assume the geographical coverage area partitioned into service areas, and define the valid scope of an item as the set of service areas where the item value is valid. To every data item sent from the server is attached its valid scope. So a data item becomes invalid when the user enters in a service area not belonging to its valid scope. However, as noted by (Kubach and Rothermel 2001), caching never speeds up the first access to an information item, and caching location-dependent data is not beneficial if the user does not return frequently to previously visited locations. Their simulation results have proven that hoarding gives better results than caching in location-dependent systems, although it is assumed that memory available for caching is unlimited and no information is removed from the cache.

Hoarding is the process of predicting the information that the user will request, for transferring it in advance to the client cache. So, the future user's query will be satisfied by the client, although the response contains a data item that has never been requested before. Several hoarding techniques have been proposed in the literature. The first proposed methods were requiring user intervention, making the system less convivial and are useless in systems where the user does not know in advance what kind of information he will need. Automated hoarding is the process of predicting the hoard set without user intervention. Kuening and Popek (Kuenning and Popek 1997) propose an automated hoarding method where a measure called semantic distance between files is used to feed a clustering algorithm that selects the files that should be hoarded. Saygin et al (Saygin et al. 2000) propose another method based on data mining techniques. This

latter uses association rules for determining the information that should be hoarded. Khushraj et al (Khushraj et al. 2002) propose a hoarding and reintegration strategy substituting whole file transfers between the client and the server, by transferring only the changes between their different versions. These changes are patched to the old version once in the destination machine. This incremental hoarding and reintegration mechanism is built within the Coda file system (Coda Group), based on the Revision Control System (RCS) (Tichy1985). Cao (Cao 2002) propose a method that allows making a compromise between hoarding and available power resources.

None of these methods deal with the spatial property of location-dependent systems. De Nitto et al (De Nitto et al. 1998) propose a model to evaluate the effectiveness of hoarding strategies for context aware systems, based on cost measures. They apply these measures to some idealised motion models. For the two-dimensional model, the area is divided in adjacent polygons. The distance between two polygons is the number of polygons that must be traversed to pass from the first polygon to the other one. The ring *k* is defined as the set of all polygons whose distance from a given polygon is equal to *k*. All the information associated to rings *0, 1, ..., k* around the starting position is hoarded. The next hoard does not occur until the user enters a polygon outside the circle *k*. One drawback of this strategy is that a lot of unnecessary information is hoarded because the user's direction is not taken into account (all the information associated with the area behind the user is useless). Another drawback is that the hoard does not begin until the user is out of the hoarded circle. So hoard misses will occur until the information related with the new user's circle is hoarded. Kubach and Rothermel (Kubach and Rothermel 2001) propose a hoarding mechanism for location-dependent systems based on infostations (Badrinath et al. 1996). When the user is near an infostation the information related to its area is hoarded. An access probability table is maintained where each data item is associated with the average probability that it will be requested. Only a fixed number of the first data items with the highest probabilities are hoarded for the purpose of not wasting bandwidth and the user's device resources. No explication is given about what a data item is. We find that a fixed number *m* is not a good criteria for this purpose in a realistic system because what ever data items can be (files, tables, fields in a table, web pages, real word entities…) there will be always differences in the memory required for them, so a fixed number of data items can always exceed the space reserved for them. When no assumptions can be made on the future user's movement, all the information related to the infostation area is hoarded. So this mechanism is not adapted for systems managing a huge amount of data.

3 Proposed Method: Spatial Hoarding

3.1 Method Description

In pervasive location-dependent systems, the majority of the user's queries are related to the area where he is. The problem of latency is crucial for local queries (Zheng et al. 2002) whereas for non-local queries user movement for a short time does not invalidate the response. So our mechanism must hoard the information related to the current position of the user. As mobile devices usually suffer from limited storage and memory capacities, we cannot hoard the information associated to a big area. We consider the space divided in adjacent squares. We use the Peano encoding with N-ordering (Laurini and Thompson 1992, Moktel et al. 2003) for its known advantage of stability (we can add an other area to the area covered by the system without affecting the encoding), its usefulness for indexing spatial databases and because it allows the use of the Peano algebra for solving spatial queries. In the following we will call a square a square of length 2 and a sub-square a square of length 1 (see Fig. 1). The curve in the figure shows the path of the user. The purpose of the method is to hoard the information that the user will the most probably access. The user after leaving the square, in which he is located, will be in one of the eight adjacent squares. But hoarding all the squares will waste resources with unnecessary information because the user direction is not taken into account. For the purpose of exploiting user's orientation, we choose to make the hoarding decision when the user enters in a new sub-square. This way, if the user leaves his current square, he will be in one of the three squares adjacent to his sub-square. Thus, dividing squares in sub-squares and making the hoarding decision in the sub-squares boundaries, allows us to take the user's direction into account and to restrict the number of squares to hoard from eight to only three. Usually, one or two of the adjacent squares are already in the client and only the remaining one or two squares will be hoarded. As discussed above a caching invalidation scheme is also necessary for freeing memory for more accurate data. The user's location is also used as an invalidation criterion, and we invalidate all the squares that are not adjacent to the user's square. We summarise this as follows:

- When the user enters in a new sub-square, its three adjacent squares must be hoarded.
- The information located in the squares non adjacent to the user's square must be dropped.

As the spatial property is used both as a hoarding and a cache invalidation criterion, we call this method "*Spatial Hoarding"* (SH).

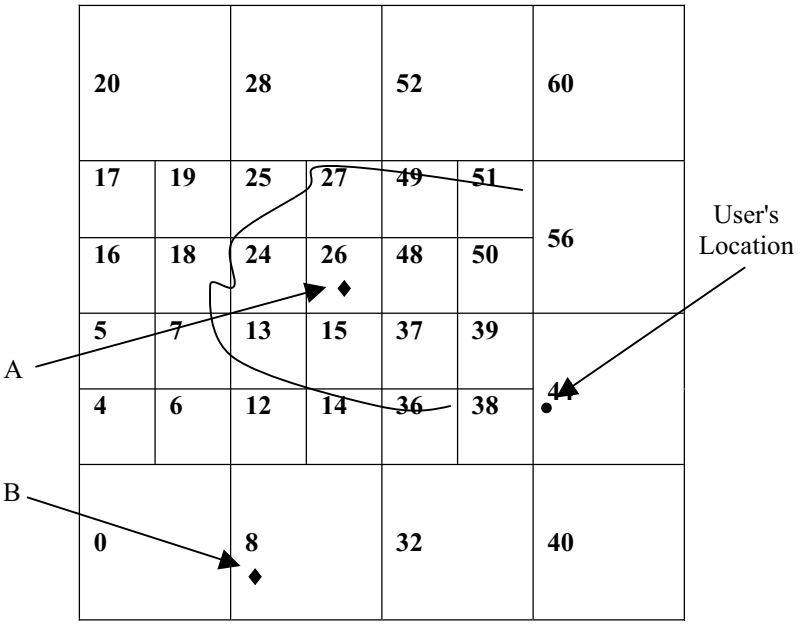

Fig. 1 User's route in an area divided into adjacent peano N-ordered squares

Table 1 summarises how the Spatial Hoarding (SH) method is applied to the user's path portion of figure 1. When the user is in the sub-square 36, the squares 8, 12, 32, 36 are available in the client's cache. When he enters to the sub-square 14, the hoarding and the cache invalidation criteria are checked to decide which squares to hoard and which one to delete. The three adjacent squares of the sub-square 14 are 8, 32 and 36, which are already in the client's cache. So no hoard is needed. For the cache invalidation criteria, there are no non-adjacent squares to the square 12, so no deletion is needed. Then, the user moves to the sub-square 12, which is adjacent to the squares 0, 4 and 8. Squares 0 and 4 are not available in the client's cache, so they will be hoarded. For the cache invalidation criteria, here again there are no non-adjacent squares to the square 12. The cache invalidation criterion is not satisfied until the user reaches the sub-square 7. 7 is a sub-square of square 4. The adjacent squares of square 4 are 0, 8, 12, 16, and 24. As Squares 32 and 36 are in the client's cache and are not adjacent to the square 4, they are invalidated by the cache invalidation criteria and are dropped.

We give an algorithm of the SH method in section 4.

Table 1. The Spatial Hoarding method applied to the user's path of Fig. 1

Sub-squares	Available squares	Squares to hoard	Squares to drop
36	8, 12, 32, 36		
14	8, 12, 32, 36		
12	8, 12, 32, 36	0, 4	
13	0, 4, 8, 12, 32, 36	16, 24	
7	0, 4, 8, 12, 16, 24, 32, 36		32, 36
18	0, 4, 8, 12, 16, 24		0, 8
19	4, 12, 16, 24	20, 28	
25	4, 12, 16, 20, 24, 28		
27	4, 12, 16, 20, 24, 28	48, 52	
49	4, 12, 16, 20, 24, 28, 48, 52		4, 16, 20
51	12, 24, 28, 48, 52	56, 60	

3.2 Client's Capability

Let's consider again the scenario of a tourist looking for the nearest restaurant. In Figure 1, the tourist is in the sub-square number 14, there is a restaurant A in the sub-square 26 and a restaurant B in the sub-square 8. If the client answers the query of the nearest restaurant, the response will be restaurant B, although restaurant A is nearer, because the sub-square 26 is not available in the client.

We define the capability of the client as the area where the client is able to give a correct answer to a query of this kind. The client's capability is the area delimited by the circle whose centre is the current user's location and whose radius is the distance between the user and the nearest vertex of the polygon delimiting the squares available on the client. For our example, this vertex is the upper side of the square 12. So, before giving an answer to the user, the client must look for the nearest restaurant within its capability circle. If no restaurant is found, the query must be transferred to the server.

3.3 Operating in Disconnected Mode

Another limit to mobile devices is their low power capacity. The SH method applied to the example of Figure 1 shows that hoarding is needed only in 5 sub-squares of the 11 ones traversed. We can exploit this by allowing the user to operate in doze or disconnected mode for saving power consumption. The application interface must allow the user to know when he can switch to disconnected mode and when he must reconnect.

4 Implementation

4.1 Algorithm

We model the available squares in the client's cache by a linked list whose nodes are squares having an attribute where we store its corresponding Peano key.

Algorithm 1 is the algorithm implementing the cache invalidation and the hoarding operations. Algorithm 2 is the algorithm retrieving the 3 adjacent squares to a given sub-square.

```
Algorithm 1: Application of the cache invalidation
and the hoarding criteria's
  Input: List of the available squares (list),
         Array of the adjacent squares to the
               current square (T1[9]),
         Array of the adjacent squares to the
               current sub-square (T2[3])
  Procedure:
     Integer i;
     /*Cache invalidation criteria*/
     temp := list.first;
     temp2 := new(square);
     while ((notin(temp, T1) = = true) and (temp
            != NULL) do
        first := temp.next;
        temp := first;
     end while
     while (temp != NULL) do
        if (temp.next != NULL) then
```

```
          if (notin(temp.next, T1) = = true) then
             begin
                temp2 := temp.next;
                temp.next := temp2.next;
          if (tmp2 = = last) then
             last = tmp;
    end if
                free(temp);
             end
          else then
                temp := temp.next;
          end if
     else then
        temp := temp.next;
     end if
  end while
  /* Hoarding Criteria*/
  for i :=  1 to T2.length do
     if (isnotin(T2[i], list) = = true) then
        add(list, T2[i]);
  end for
```

The algorithm begins with the application of the cache invalidation criteria. The first element of the list is treated alone because it has no previous node pointing to it. Each node of the list is compared, using the function "notin" with the array T1; if a node (square) does not exist in the array T1 (if this square is not adjacent to the current square), it is dropped from the list. Then, the hoarding criterion is applied. Each element of T2 (array of the adjacent squares to the current sub-square) is compared to the list using the function "isnotin". Every element of T2, which does not exist in the list, is added to the latter using the function "add".

Algorithm 2: Determination of the adjacent squares to the current sub-square.

```
  Input: Peano key of the current sub-square (P)
  Output: The array of the 3 adjacent Squares T[3]
  Procedure:
     integer T[3];
     integer A[3][2];
     integer x, y, i;
     x := get_x(P);
     y :=get_y(P);
```

```
switch (P mod 4)
begin
  case 0 : A[1][1] := x – 1; A[1][2] := y;
           A[2][1] := x – 1; A[2][2] := y - 1;
           A[3][1] := x ; A[3][2] := y - 1;
           Break;
  case 1 : A[1][1] := x; A[1][2] := y + 1;
           A[2][1] := x – 1; A[2][2] := y + 1;
           A[3][1] := x - 1; A[3][2] := y;
           Break;
  case 2 : A[1][1] := x; A[1][2] := y - 1;
           A[2][1] := x + 1; A[2][2] := y - 1;
           A[3][1] := x  + 1; A[3][2] := y;
           Break;
  case 3 : A[1][1] := x; A[1][2] := y + 1;
           A[2][1] := x + 1; A[2][2] := y + 1;
           A[3][1] := x +1; A[3][2] := y;
           Break;
end
for i :=  1 to 3 do
   T[i] := get_p(A[i][1],A[i][2]);
   T[i] := T[i] – (T[i] mod 4)
end for
output T
```

First, the coordinates x and y are deducted from the Peano key P using the functions “get_x” and “get_y”. Then, the position of the current sub-square in its parent square is determined, because the determination of the adjacent sub-squares depends on it. After determining the coordinates of the adjacent sub-squares, the corresponding Peano keys are deducted using the function “get_p”. Then, the number of the left bottom sub-square is deducted, because the number of this sub-square added to a length 2 is the name of the square.

We do not give the algorithm of the determination of the adjacent squares to the current square, because it is quite similar to algorithm 2.

4.2 What to Hoard

Location-dependent systems such as tourist guides, as noted before, can use different kinds of data (text, graphics, maps, images, audio clips, film clips…). So the amount of the multimedia data managed is very important.

We have proposed to divide the area covered by the system into adjacent squares, because hoarding the information related to all the area cannot be implemented in a real system, because of the limited memory and storage capacity in the user's device. Depending on the user's device resources and the available bandwidth, a fixed amount of space A will be reserved to the client's cache. As the maximal number of squares that can be available in the client's cache is 9, the amount of hoarded data cannot exceed A/9 in each square. As we deal here with systems that manage a huge amount of data, the amount of information related to some squares can exceed the A/9 value. Also, some data items associated to a square may have low probabilities of access. So transferring this data may only waste resources. Kubach and Rothermel (Kubach and Rothermel 2001) have described how to maintain access probability tables, where each data item is associated with the average probability that it will be requested, and propose to hoard only a fixed number of the first elements. We think that, a fixed number of data items is not a criteria that can be implemented in practice, because data items do not require the same space in memory, so hoarding a fixed number of data items can exceed the space reserved for the cache. In the following we will determine what we mean by a data item within the SH method, we will explain how we will be able to add and drop data items dynamically depending on the amount of space available on the client's cache and how to retrieve all the information related to a given square.

Laurini and Thompson (Laurini and Thompson 1992) have generalised the concept of hypertext to hyperdocument where the content portions can always be displayed on a screen or presented via a loudspeaker, and define the hypermap as multimedia documents with a geographic coordinate based access via mouse clicking or its equivalent. Database entities are represented by graphical means, and clicking a reference word or picture allows the user to go to another node. They present the following relational model:

```
WEB(Document_node-ID, (Word_locator
(To_document_node-ID, Type_of_link)*)*,
(From_document_node-ID, Type_of_link)*)
```

Where * indicates the nesting (the data nested are in a separate table; however, they may be stored as a long list in the parent table).

- Type_of_link refers to the nature of the path from one node to another.
- Word_locator is the word, graph unit, or pixel, or other element, in the first element.

They explain also how to deal with spatial queries for retrieving hypermap nodes. By Peano relations the solution is:

```
Document(Document_node-ID,          (Peano_key,
Side_length)*)
```

Within the SH method we will consider document nodes as the data items to which the average access probability will be associated:

```
Document(Document_node-ID,   P(Document_node-ID),
(Peano_key, Side_length)*)
```

The first operation is to retrieve all the documents related to a candidate square for hoarding, sorted by decreasing order of their average probability of access. Then, the application will use the operating system and DBMS primitives for associating each document to its amount of space in memory. The documents will be added to the hoard set until the maximum value fixed for their square is reached. This value can be exceeded if there is sufficient space in the client. As noted before, the client can drop the data items with the lower probabilities later if necessary.

Every click on a node or a document consultation will be kept in a log file in the client, and will be sent later to the server for updating the average access probability for each document node.

5 Future Work

We are in the final stages of the development of a simulation prototype for the Spatial Hoarding method. Our preliminary simulation results show that the SH policy improves the cache hit ratio and reduces significantly the query latency.

When there is an important number of user's in a given area, the information related to the same square may be hoarded several times for different clients. In our future work we will focus on the issue of saving the bandwidth in the case of multiple users.

6 Conclusion

We have presented an innovative policy for resolving the problem of latency in location-dependent systems. Our mechanism makes no assumptions about the future user's movement and thus deals with the complexity of real world applications. We proposed solutions to all the problems related to the method's implementation in an elaborated spatial information system managing multimedia information

Our hoarding mechanism improves the cache hit ratio, thus reduces the uplink requests, and reduces the query latency. Compared to the previous hoarding mechanisms whose main aim was to allow disconnected information access, but making the user in the risk of accessing obsolete informa-

tion, our method allows the user to have access to the most recent data. Using the user's position as a cache invalidation criterion reduces the need for extra communication between the client and the server for checking cache consistency.

References

Badrinath, B.R., Imielinsky, T., Frankiel, R. and Goodman, D., 1996. Nimble: Many-time, many-where communication support for information systems in highly mobile and wireless environments, http://www.cs.rutgers.edu/~badri/dataman/nimble/.

Cao, G., 2002, Proactive power-aware cache management for mobile computing systems. IEEE Transactions on computers 51, 6, 608-621.

Cheverst, K., Davis, N., Mitchell, K., Friday and Efstriatou C., 2000, Developing a context-aware electronic tourist guide: some issues and experiences. In Proceedings of CHI'2000, Netherlands, pp. 17-24.

The Coda Group, Coda file system, http://www.coda.cs.cmu.edu/.

De Nitto, V.P., Grassi, V., Morlupi, A., 1998, Modeling and evaluation of prefetching policies for context-aware information services. In Proceedings of the 4th Annual International Conference on Mobile Computing and Networking, (Dallas, Texas, USA), pp. 55-64.

Khushraj, A., Helal, A., Zhang, J., 2002, Incremental hoarding and reintegration in mobile environments. In Proceedings of the International Symposium on Applications and the Internet (Nara, Japan).

Kubach, U., and Rothermel, K., 2001, Exploiting location information for infostation-based hoarding. In Proceedings of the 7th International Conference on Mobile Computing and Networking (New York, ACM Press), pp. 15-27.

Kuenning, G. H., and Popek, G. J., 1997, Automated hoarding for mobile computers. In Proceedings of the 16th ACM Symposium on Operating Systems Principles, (St. Malo, France), pp. 264-275.

Laurini, R., and Thompson, A.D., 1992, Fundamentals of Spatial Information Systems (A.P.I.C. Series, Academic Press, New York, NY).

Lee, D.L., Lee, W.C., Xu, J., and Zheng, B., 2002, Data management in location-dependent information services. IEEE Pervasive Computing, 1, 3, 65-72.

Abowd, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R., Pinkerton, M., 1997, Cyberguide: a mobile context-aware tour guide. Wireless Networks 3, 5, pp. 421-433.

Malaka, R., 1999, Deep Map: the multilingual tourist guide. In Proceedings of the C-STAR workshop.

Mokbel M-F, Aref W-G., Kamel I.: Analysis of Multi-Dimensional Space-Filling Curves. GeoInformatica 7(3): 179-209 (2003)

Poslad, S., Laamanen, H., Malaka, R., Nick, A., Buckle, P., and Zipf, A., 2001, CRUMPET: Creation of user-friendly mobile services personalised for tour-

ism. In Second International Conference on 3G Mobile Communication Technologies (London UK), pp. 28-32.

Saygin, Y., Ulusoy, Ö., and Elmagarmid, A.K., 2000, Association rules for supporting hoarding in mobile computing environments. In Proceedings of the 10th International Workshop on Research Issues in Data Engineering (IEEE Computer Society Press).

Tichy, W.F., 1985, RCS - A system for version control. Software-Practice and Experience, 15, 7, pp. 637-654.

Xu, J., Tang, X., Lee, D.L., and Hu, Q., 1999, Cache coherency in location-dependent information services for mobile environments. In Proceedings of the 1st International Conference on Mobile Data Acess (Springer, Heidelberg, Germany), pp. 182-193.

Zarikas, V., Papatzanis, G., and Stephanidis, C., 2001, An architecture for a self-adapting information system for tourists. In Proceedings of the 2001 workshop on Multiple User Interfaces over the Internet, http://cs.concordia.ca/~seffah/ihm2001/papers/zarikas.pdf.

Zheng, B., Xu, J., and Lee, D.L., 2002, Cache invalidation and replacement strategies for location-dependent data in mobile environments. IEEE Transactions on Computers 51, 10, pp. 1141-1153.