# Evaluation of an Architecture for the Distributed Processing of Continuous Location-Dependent Queries[⋆]

S. Ilarri[1⋆⋆], E. Mena[1], and A. Illarramendi[2]

[1] IIS Dept, Univ. of Zaragoza, Maria de Luna 1, 50018 Zaragoza, Spain
{silarri, emena}@unizar.es
[2] LSI Dept, Univ. of the Basque Country, Apdo. 649, 20080 Donostia, Spain
jipileca@si.ehu.es

**Abstract.** With the current advances of mobile computing technology, we are witnessing an explosion in the development of applications that provide mobile users with a wide range of services. Of special interest are those applications that exploit the particular features of mobile environments to provide the user with context-aware information. In particular, we focus on location-dependent queries, which are still a subject of research mainly due to the lack of an architecture that is well-adapted to deal with continuous location queries in an efficient way.
In this paper we present the distributed architecture that we propose to process continuous location-dependent queries in mobile environments. It is based on a network of mobile agents that adapt themselves to the movements of the objects involved in a query, with the goal of tracking their locations and keeping the query answer up-to-date. The main goal is to minimize the wireless communications and the overload of the user device. We evaluate our proposal, showing that the system achieves a good precision and scales up well.

**Keywords:** location-dependent querying, tracking moving objects, continuous querying

## 1 Introduction

With the current advances of mobile computing technology, we are witnessing an explosion in the development of applications that provide mobile users with a wide range of services. While most of these services are the counterpart of those available in desktop computers, there exist other applications that exploit the special features of mobile environments to supply more relevant information. For example, the answer to a location-dependent query depends on the locations of

⋆ This work was supported by the CICYT project TIC2001-0660, the DGA project P084/2001, and the research institute I3A.

⋆⋆ Work supported by the Aragón Government and the European Social Fund (ref. B132/2002).

moving objects. A sample location-dependent query is "show me the available taxi cabs within three miles of my current location", which could be very useful, for example, for a user looking for an available taxi cab while walking home in a rainy day [12].

Nowadays there exist commercial applications that deal with some location-dependent aspects. However, they present many disadvantages, among which we would like to point out the lack of a general architecture that is well adapted to deal with continuous queries in an efficient way. This implies that current solutions are only suitable for the specific context for which they were developed and they do not work in a global, large-scale and heterogeneous environment.

In this paper we present first the distributed architecture that we propose to process continuous location-dependent queries in mobile environments. Mobile agents [9], which have been proved to be very suitable for mobile computing applications, are the basis of our architecture. Thus, the system deploys a network of agents that adapt themselves to the movements of the objects involved in a query. The goal is to track the locations of such objects and keep the query answer up-to-date with the minimum cost of wireless communications and overload for the user device. We then evaluate our proposal, showing that the system achieves a good precision and that the architecture is scalable.

The rest of the paper is as follows. In Section 2 we describe the distributed architecture that we propose for the processing of location-dependent queries. The empirical evaluation of our query processing approach appears in Section 3. In Section 4 we present some related works. Finally, conclusions and future work are included in Section 5.

## 2 Architecture for the Processing of Location Queries

In this section, we describe first the infrastructure that we consider for the processing of location-dependent queries, then our query processor approach and, finally, the main reasons for using mobile agent technology.

### 2.1 Underlying Infrastructure

As framework of our work, we consider an infrastructure composed of moving objects and proxy computers that provide location information.

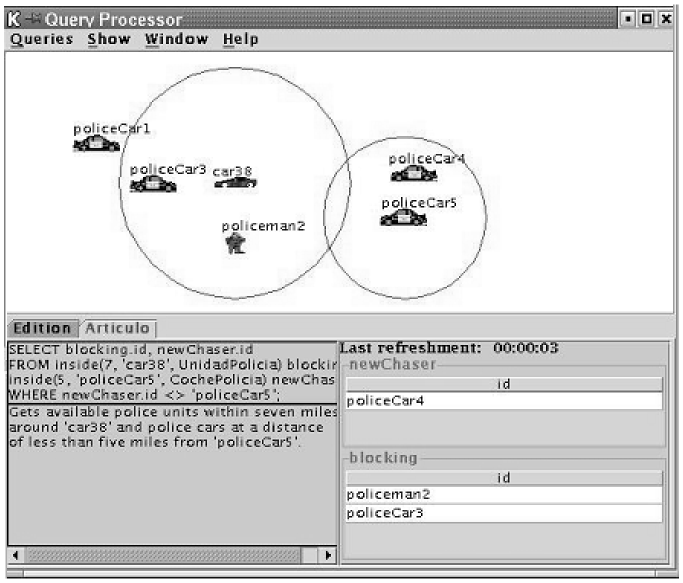
*Moving objects* are entities provided with a wireless device (e.g., a car or a person with a wireless communication device) and a mechanism that obtains their location. Our architecture works well for positioning methods with different precision [3], whether it is based on GPS or it requires network assistance.

*Proxies* are computers that provide location information about objects located within a certain area, that we call *proxy area*. In a cellular network [10, 2], a proxy area is obtained as the union of the coverage areas of one or more base stations. Although our concept of proxy is similar to that of *Location Server* [7], we decided to use a different term to fit it to the specifics of our decentralized query processing approach: 1) they provide the software infrastructure needed for the execution of mobile agents, and 2) there is a module called *Data Management System* (*DMS*) at the proxy.

The DMS handles location data about moving objects within its proxy area in order to answer SQL queries about them. We do not focus here on the problem of how the DMS is aware of the location of moving objects, but we would like to highlight some interesting alternatives: 1) it could receive GPS data from moving objects and use a DOMINO database [13] to update efficiently these data, 2) it could use data stream technology [1] to process the incoming location data regarding the queries currently submitted to the proxy, 3) it could obtain the location data by using a network-based positioning method, 4) it could query the moving object themselves about their location, or 5) any hybrid approach is also possible.

## 2.2 Query Processing Approach

We explain in this section our approach for the processing of location-dependent queries [5]. For that purpose, we introduce first some terminology related to location-dependent queries, and then we show the general query processing algorithm and the (mobile) agent-based architecture that we use to process the queries in a distributed way.



```
SELECT blocking.id, newChaser.id
FROM inside(7, 'car38', UnidadPolicia) blockir
inside(5, 'policeCar5', CochePolicia) newChas
WHERE newChaser.id <> 'policeCar5';
```

Gets available police units within seven miles around 'car38' and police cars at a distance of less than five miles from 'policeCar5'.

**Last refreshment: 00:00:03**

newChaser

| id |
|---|
| policeCar4 |

blocking

| id |
|---|
| policeman2 |
| policeCar3 |

**Fig. 1.** Example of location-dependent query running on our prototype

Location queries are composed of *location constraints* that express location-dependent conditions that moving objects must satisfy to be in the answer to the query; e.g., the constraint "Inside(5 miles, 'car38', policeCar)" selects the police cars (the *target class*, whose instances are termed *target objects*) within five miles around the moving object identified as "car38" (the *reference object*). We show in Figure 1 a snapshot of our application for a more complex sample query.

```
 1: location-dependent query $\Rightarrow$ parametrized SQL queries
 2: while (not cancel) do
 3:   Obtain the locations of the reference objects
 4:   Update the SQL queries with the locations of the reference objects
 5:   for each SQL query do
 6:     execute the query
 7:     if there are several SQL queries about the same target class then
 8:       Join the result of such queries
 9:     end if
10:   end for
11:   Present the answer to the user
12: end while
```

**Fig. 2.** Basic algorithm for location query processing

In Figure 2 we present the basic algorithm to obtain an answer to a location query. The query is transformed into standard SQL queries about the locations of target objects. These queries are parametrized with the location of the reference objects, that must be consequently obtained before executing the queries. We will explain in the following how this algorithm is executed in a distributed way; thus, for example, the line 6 in Figure 2 involves several proxy computers. The steps performed in our distributed query processing approach are:

1. *Analysis of the user query and translation into SQL queries* (line 1 in Figure 2). The *Query Processor* is the application executed on the user device (that we call *monitor*) that allows the user to issue queries. It first transforms the user query into an intermediate specification. Specifically, it obtains, for each constraint in the query, its *reference object*, its *target class*, and an *area of interest*. Then, from this representation, one SQL query is obtained for each constraint, which is parametrized with the location of the reference object.
2. *Deployment of a network of agents*, that consists of three steps (see Figure 3):
   - The Query Processor sends a *MonitorTracker agent* to the proxy in charge of the monitor's location (step 1 in Figure 3). This agent performs three main tasks: a) to follow the monitor wherever it goes (moving from proxy to proxy), b) to store the data requested by the user in case of disconnection of the monitor, and c) to refresh the data presented to the user in an efficient manner, minimizing the wireless communications with the monitor (it will send only the necessary information).

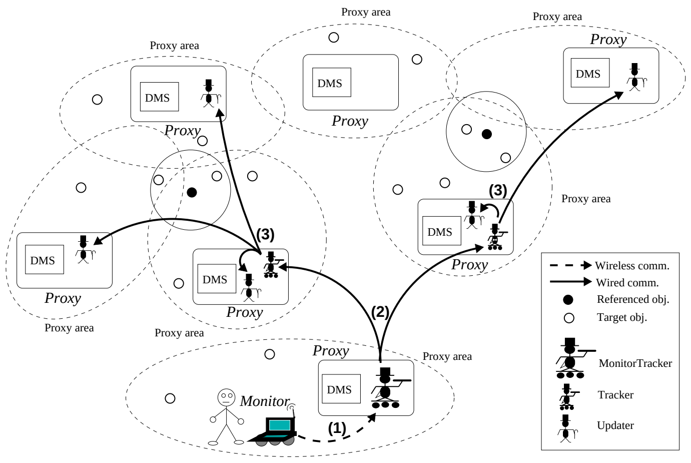


**Fig. 3.** Deployment of the network of agents

- For its last task, the MonitorTracker sends a *Tracker agent* to the proxy that manages the location information of each reference object, initialized with the SQL queries related to such a reference object (step 2 in Figure 3). A Tracker agent performs three main tasks: a) to keep itself "close" to the proxy that manages the location of its reference object, traveling from proxy to proxy when needed in order to minimize the communication costs of tracking the reference object, b) to detect and process new locations of its reference object by querying the corresponding DMS at its proxy (line 3 in Figure 2), and c) to detect and process the location of target objects inside the area of interest around its tracked reference object.
- For its third task, a Tracker agent obtains for each SQL query in which its reference object appears, the proxies whose area intersects the area of interest of such an SQL query. Then, it creates one *Updater agent* on each of these proxies (step 3 in Figure 3). The Updaters are static agents whose goal is to retrieve relevant target objects from the DMS at their proxy. They are initialized with the SQL query that they must execute; such query is based on the current location of the reference object[1]

3. *Execution of standard queries* (line 6 in Figure 2). Each Updater agent executes its SQL query against the DMS of the proxy where it resides, with the goal of retrieving the location of relevant target objects and communicate the necessary data to its Tracker agent.
4. *Obtaining an answer.* Data obtained by Updater agents are sent to its corresponding Tracker agent. The Tracker agent performs the union of the results obtained by its Updater agents. Each Tracker agent sends its results to its MonitorTracker, which combines the results coming from the different Track-

[1] Such location parameters will be updated by the Tracker whenever necessary.

ers (lines 7-9 in Figure 2) and sends the final answer to the monitor. Finally, the monitor presents the answer to the user (line 11 in Figure 2).

Notice in Figure 3 that the only wireless data transfer occurs when the Query Processor sends the MonitorTracker to the proxy of the monitor and when the MonitorTracker sends a new answer to the Query Processor: any other communication occurs among proxies using a fixed network.

5. *Keeping the answer up-to-date.* The proposed architecture have been designed with the goal of processing continuous location-dependent queries efficiently. Continuous queries must be refreshed with a certain *refreshment frequency* until they are explicitly canceled by the user.

   While a continuous query is executing, the deployed network of agents must adapt to changes in the locations of moving objects. For example, as a reference object moves, its associated area of interest moves too. Therefore, the related Tracker agent must rearrange its network of Updater agents and update their SQL queries with the new location of the reference object (line 4 in Figure 2). Similarly, if a reference object disappears from its current proxy[2], the Tracker agent needs to identify the new proxy in charge of the location of the reference object. Besides, the Tracker will also travel there, with two purposes: 1) remote requests about the location of the reference object are avoided, and 2) when the reference object changes of proxy it will be easier for the Tracker to locate its new proxy, as most location databases schemes are more efficient when the searches imply nearby proxies[10].

   It is not the purpose of this paper to describe how the agents synchronize in order to provide timely answers (see [6] for more details of our approach).

### 2.3 Use of Mobile Agents

We have decided to use mobile agents technology because it provides us with suitable mechanisms to solve the presented problem in a distributed, efficient and convenient manner. Of course, we could have used an RPC mechanism instead. However, this would lead to a non-flexible and more difficult to implement solution: agent migration would be replaced by remote invocations that create and destroy threads that represent the behavior of our agents. Mobile agents allow us to bring the required functionality to any computer, instead of installing and launching specialized daemons/servers on each machine to provide the wanted services. Besides, mobile agents ease the addition of new services once the system is working: new agents with new functionalities could travel to the user device or proxies without re-installing anything there. Finally, it has been proved that the overall performance of a system based on mobile agents is, at worst, as good as that of the equivalent system based on remote communications [8].

[2] This is detected by the Tracker agent because the current proxy cannot provide the location of the reference object anymore.

## 3 Experiments

In this section, we describe some tests that we have performed to check the reliability, accuracy and scalability of our prototype, that we have also modelled using Petri nets. The prototype has been developed using Java as programming language and Grasshopper [11] as mobile agent system. We consider a scenario composed of six proxies; six Pentium IV 1.7 GHz with Linux have been used to play the role of such proxies. DMSs have been implemented as MySql databases, one on each proxy.

In the following, we present the behavior of the system when tracking a single object, the accuracy of the query processor when executing a continuous location query that retrieves a set of objects, and how the system scales up when the number of concurrent location queries increases.

### 3.1 Tracking the Location of a Single Moving Object

We want to show here the precision of the answers presented to the user with respect to the real situation. We consider an object moving at 5 du/sec (distance units per second; e.g., 1 du = 1 mile or 1 du = 1 meter).

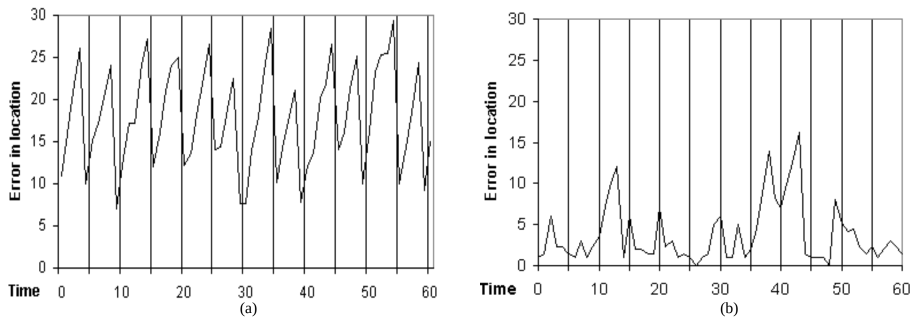


**Fig. 4.** (a) Monitoring an object at 5 du/sec ); (b) using an estimator

In Figure 4.a we show the imprecision (error in distance) committed by the Query Processor for a query that simply tracks the location of such an object with a frequency of 1 answer refreshment every five seconds. We show the error in distance, i.e., the difference between the real location of the object and the location shown to the user at each time instant. Notice that the minimum imprecisions occurs every five seconds, corresponding to refreshments of the query answer, where the error is only due to the time elapsed since a location is queried until that information is presented to the user[3]. Figure 4.b was obtained by using a simple location estimator based on the last known orientation and speed of the moving object. This estimator allows us to increase the frequency of answers

[3] Moving objects are moving in the meanwhile!

while keeping the same frequency of accesses to proxies and wireless communications; in this case, we estimate one answer every second instead of refreshing the answer every five seconds. The increase in the location error committed by the query processor between seconds 35-45 in Figure 4.b is due to a wrong estimation of locations caused by unexpected changes of the direction of the moving object. We have tested the system with different speeds for the moving object and observed, in all the cases, an improvement in the results when using an estimator.

### 3.2 Real Answer vs. Obtained Answer

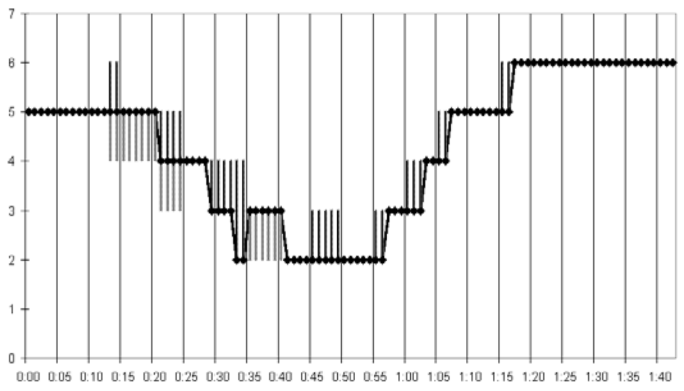


**Fig. 5.** Accuracy in the answer presented to the user

In Figure 5 we evaluate the query processor in terms of expected answers, considering a refreshment period of five seconds and an object speed of three du/sec. The X-axis shows several time instants (continuous vertical lines represent refreshments of the answer) and the Y-axis indicates a number of objects. The continuous dark line indicates the number of objects in the ideal answer at each moment. Short vertical lines mean how many extra objects (lines above the continuous line) and missing objects (lines below the continuous line) were obtained by the query processor as answer to the issued query. For instance, between 0:35 and 0:40 there is one missing object in the answer presented to the user, and between 1:00 and 1:03 the answer shown by the query processor is presenting one extra object. Thus, short vertical lines represent temporal imprecision of the query processor, due to the time that it needs to detect a relevant change in the information provided by proxies. This error becomes bigger due to the movement of moving objects.

Notice that several objects can move at the same time in a way that the query processing mechanism can incur in imprecision. Please see the interval 0:45-0:50, when imprecision arises because one object exited the area of interest at 0:44 (which was detected by Updaters) and another object entered the area of interest at the same time (the number of objects in the ideal answer is still the same). However, the exiting object was not immediately detected by the system as one Tracker agent was traveling between BSs at that moment.

In Figure 6 we show how the averaged number of errors (based on extra and missing objects) per second changes with the refreshment period and the speed of involved objects.

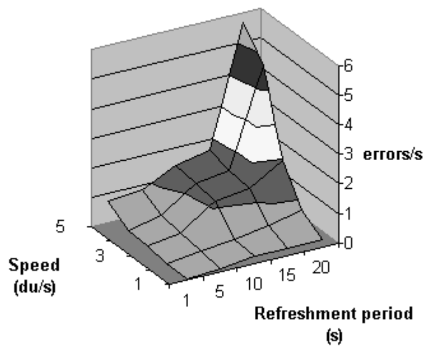


**Fig. 6.** Errors for different speeds and refreshment frequencies

We have also evaluated the query processor in terms of expected answers when issuing a query on a scenario with 50 objects that are strategically distributed in order to cause quick changes in the ideal answer, which will increase imprecision. In Figure 7 we show the performance that we get for this scenario. Notice that errors are not completely fixed by refreshments (vertical dot lines) when, by chance, a refreshment and a change in the ideal answer occurs at the same time (which happens, for example, within the interval 1:00-1:15). Another interesting comment about Figure 7 is that at time instant 00:47 one object entered the area of interest and another one left it (so the total number of objects in the ideal answer remains the same); in the next 2 seconds these errors are fixed thanks to the estimation of locations.

## 3.3 Scalability

We show in this section how the reliability of the query processor decreases when the number of (users issuing) continuous queries increases. The more queries, the more agents on the system (see Figure 8.a). However, notice that the environment is highly distributed, and therefore many queries are needed to overload a single proxy[4], as shown in Figure 8.b.

[4] Unfortunately, the agent platform used (Grasshopper) becomes unreliable for a large number of mobile agents.

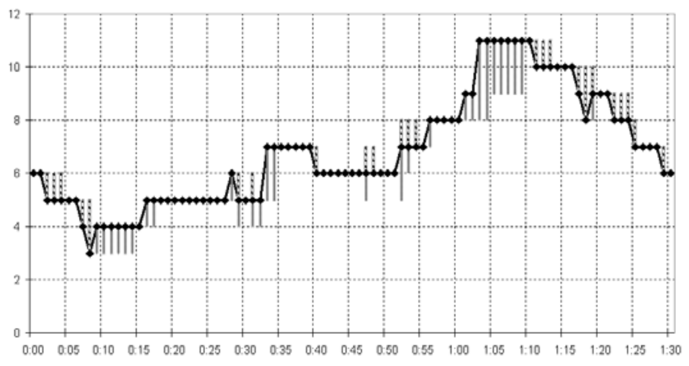


**Fig. 7.** Accuracy in the answer for a complex scenario

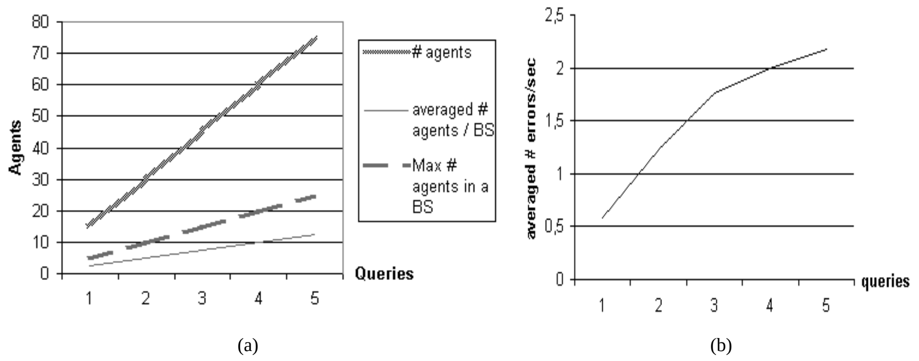


**Fig. 8.** Scalability of the Query Processor

Notice that increasing the number of moving objects does not affect the performance of our system because of two reasons: 1) The query processor is not interested in all the moving objects of the system, only in those belonging to a target class inside a certain area, 2) if the number of target objects increases, more objects could belong to the answer and therefore Updaters will retrieve more objects, but this issue does not affect the performance. Moreover, if the number of proxies increases, the system becomes more distributed but this does not negatively affect our system either (the overload at each proxy could decrease!). The real bottleneck of the system is the number of agents deployed, and this parameter only depends on the number of concurrent queries.

Apparently, we could improve the performance of the system by sharing agents among queries. For example, the same Updater could execute SQL queries concerning different location queries. However, we would need several threads, since the answers for such location queries could need to be refreshed at different time instants (e.g., the refreshment frequencies could be different). As the performance of several agents performing a single task is comparable to that of a single agent with several threads (one for each task), we do not advocate this solution.

## 4 Related Work

The DOMINO project [13] proposes a model to represent moving objects in a database. They focus on the problem of how to store location information about moving objects in a database with the goal of processing spatial and temporal queries in an efficient way. However, they consider a centralized architecture (the location data of moving objects are available at a single data repository).

In MobiEyes [4] a distributed approach for the processing of location queries is also proposed. However, in contrast to our proposal, the query processing is performed on the moving objects themselves, with the following main disadvantages: 1) moving objects must have certain “intelligence” as they must be aware of a geographic grid and be able to process queries, 2) the processing and communication load of a moving object depends on the number and type of queries issued into the system, 3) the approach is not completely distributed, as all the moving objects must communicate with a single centralized computer (the mediator), and 4) they assume that the user is not interested in a precise location of target objects.

In [14] the main concern is how to provide an up-to-date answer to the user while minimizing communications, by deciding when to transmit results to a mobile host. However, it is not well-adapted to deal with continuous queries that ask for locations of moving objects (e.g., monitoring a truck fleet). It is based on predicted paths and also considers a centralized architecture.

## 5 Conclusions and Future Work

In this paper we have presented an architecture for the processing of continuous location-dependent queries in wireless environments. We have then evaluated experimentally the performance of our prototype, showing the precision and scalability of the system. The main features of our approach are:

– We consider queries whose answer can depend on the movement of any relevant object.
– We propose a distributed architecture based on mobile agents that obtain the answer for a query by getting information from a distributed collection of DMSs with the goal of tracking relevant moving objects in an efficient way.
– No assumptions are made regarding how the locations of moving objects are obtained. Thus, our system works for different granularities of location information (e.g., GPS locations). Similarly, we do not overload mobile objects with query processing tasks.

As future work, we plan to research the advantages that data stream technology can provide in this context. Another important issue is to extend our query language (management of new spatio-temporal constraints).

## References

1. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of 21st ACM Symposium on Principles of Database Systems (PODS 2002)*, 2002.
2. D. Barbará. Mobile computing and databases - a survey. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):108–117, Jan-Feb 1999.
3. Harvey M. Deitel, Paul J. Deitel, Tem R. Nieto, and Kate Steinbuhler. *Wireless Internet and Mobile Business -How to Program-*. Prentice Hall, 2001.
4. Bugra Gedik and Ling Liu. Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In *9th Conference on Extended Database Technology (EDBT 2004), Heraklion-Crete (Greece)*, March 2004.
5. S. Ilarri, E. Mena, and A. Illarramendi. A system based on mobile agents for tracking objects in a location-dependent query processing environment. In *Twelfth International Workshop on Database and Expert Systems Applications (DEXA'2001), Fourth International Workshop Mobility in Databases and Distributed Systems (MDSS'2001), Munich (Germany)*, pages 577–581. IEEE Computer Society, ISBN 0-7695-1230-5, September 2001.
6. S. Ilarri, E. Mena, and A. Illarramendi. Dealing with continuous location-dependent queries: Just-in-time data refreshment. In *First IEEE Annual Conference on Pervasive Computing and Communications (PerCom), Dallas Fort-Worth (Texas)*, pages 279–286. IEEE Computer Society, ISBN 0-7695-1895, March 2003.
7. Location Interoperability Forum (LIF). Mobile location protocol specification. http://www.openmobilealliance.org/tech/affiliates/lif/lifindex.html, [Accessed 23 February 2004].
8. E. Mena, J.A. Royo, A. Illarramendi, and A. Goñi. Adaptable software retrieval service for wireless environments based on mobile agents. In *2002 International Conference on Wireless Networks (ICWN'02), Las Vegas, USA*, pages 116–124. CSREA Press, ISBN 1-892512-30-0, June 2002.
9. D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF, the OMG mobile agent system interoperability facility. In *Proceedings of Mobile Agents '98*, September 1998.
10. E. Pitoura and G. Samaras. Locating objects in mobile computing. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):571–592, July/August 2001.
11. IKV++ technologies. Grasshopper - a platform for mobile software agents. http://www.grasshopper.de/download/doc/GrasshopperIntroduction.pdf, , [Accessed 5 January 2004].
12. Jari Veijalainen and Mathias Weske. *Modeling Static Aspects of Mobile Electronic Commerce Environments, Chapter 7 in Advances in Mobile Commerce Technologies.* IDEA Group Publishing, 2002.
13. O. Wolfson, A. Prasad Sistla, B. Xu, J. Zhou, S. Chamberlain, Y. Yesha, and N. Rishe. Tracking moving objects using database technology in DOMINO. In *Fourth International Workshop Next Generation Information Technologies and Systems (NGITS'99), Zikhron-Yaakov, Israel. Lecture Notes in Computer Science, Vol. 1649, Springer, ISBN 3-540-66225-1*, pages 112–119, July 1999.
14. Kam yiu Lam, Özgür Ulusoy, Tony S. H. Lee, Edward Chan, and Guohui Li. An efficient method for generating location updates for processing of location-dependent continuous queries. In *Database Systems for Advanced Applications*, pages 218–225, 2001.