A System based on Mobile Agents for Tracking Objects in a Location-dependent Query Processing Environment*

Sergio Ilarri
IIS department
University of Zaragoza
Spain
silarri@prometeo.cps.unizar.es

Eduardo Mena
IIS department
University of Zaragoza
Spain
emena@posta.unizar.es

Arantza Illarramendi
LSI department
University of the Basque Country
Spain
jipileca@si.ehu.es

Abstract

Nowadays the number of mobile devices is increasing rapidly. However there is a lack of data services that can be efficiently performed in mobile computing environments. The main reason is that many well-known techniques have to be adapted to the new environment. Particularly, the processing of location-dependent queries is still a subject of research.

In this paper we introduce an approach to deal with location-dependent queries posed in mobile environments. A decentralized solution is proposed using mobile agents, which also help in the optimization of the communication effort.

1. Introduction

Mobile computing introduces new challenges for data management [7, 8]. In this paper we present a data service of the ANTARCTICA system [12] whose goal is the processing of location-dependent queries, i.e., queries for which the answer depends on location of objects. We deal with contexts where not only the user issuing the query can change her/his position, but the objects involved in the query can move as well. A sample location-dependent query is "find the teller machines inside a radius of three miles accepting VISA cards and their distances to my current position".

Moreover, in the considered context we deal with continuous queries (also called contiguous queries [6] and active queries [10]), i.e., queries whose answer must be updated continuously, as opposite to instantaneous queries (also called snapshot queries in [10]) for which only a single answer is obtained. We think that instantaneous queries are not very useful in a mobile environment since the answer presented to the user can become obsolete in a short time. Continuous queries introduce new problems in the query processing because the movement of any involved object could change the query answer. Moreover, we cannot afford to consider a continuous query as a sequence of instantaneous queries that are re-sent continuously to the data server. It is necessary an approach to assure updated data but optimizing communications.

As framework of our work, we consider the generally accepted mobile environment architecture, where a mobile host communicates with its base station (BS), which provides service to all the mobile hosts within its coverage area [9, 2]. We suppose that there exists a data repository on each BS, that stores information about the mobile hosts under its coverage area.

A Motivating Example

We present in the following a sample location-dependent query that we use through the paper to introduce our query processing approach:

```
SELECT   Constraint-1.id, Constraint-2.id
FROM     inside_7('car38', policeUnit)) Constraint-1,
         inside_5('policeCar5', policeCar) Constraint-2
WHERE    not(Constraint-1.busy) and Constraint-2.id<>'policeCar5'
```

This query retrieves the available police units[1] that are within seven miles around 'car38' (a stolen car), and the police cars within five miles around 'policeCar5' (the current chaser police car)[2].

The rest of the paper is as follows. Section 2 introduces some definitions and the goal of the main modules in

*This work was supported by MoviStar (a Spanish cellular phone company).

[1] By police units we mean police stations, policemen, and police cars.

[2] 'Constraint-1' is an alias that identifies the police units close to 'car38'; 'Constraint-2' identifies the police cars that can assist the chaser police car.

the architecture proposed to deal with location-dependent queries. Section 3 provides a brief description of the kind of queries that the system deals with. Section 4 presents the query processing approach and the technique used to track the objects relevant for a query. Section 5 describes some related work. Finally, conclusions and future work are included in Section 6.

2. Architecture

In this section we provide a brief description of the main modules of the system and how they interact. We begin with some interesting notations:

- *Object*: mobile or fixed device in the scenario. Objects are classified into classes of objects (with similar features). Classes are organized in a hierarchy.
- *Monitor*: the object that poses queries. It is the device of the user interested in querying the system.
- For each location-dependent constraint of a query, the following definitions are managed:

  - *Referenced objects*: objects that are the reference of the constraint. In the sample query, there exist two referenced objects: 'car38' for the constraint "inside_7('car38', policeUnit)", and 'policeCar5' for the constraint "inside_5('policeCar5', policeCar)".

  - *Target class*: the class of objects that is the target of the constraint. In the sample query, there exist two target classes: police units for the constraint "inside_7('car38', policeUnit)", and police cars for the constraint "inside_5('policeCar5', policeCar)". The instances of a target class are called *target objects*.

  The *relevant objects* of a constraint are both the referenced and target objects involved in such a constraint.

In the following we introduce the main modules involved in the proposed query processing approach. See Section 4 for a more detailed description.

- *Query Processor*. It is the application, executed on the monitor, that allows the user to pose queries.
- *MonitorTracker agent*. It is a mobile agent residing on the BS that provides coverage to the monitor. It performs three main tasks: 1) to follow the monitor wherever it goes (moving from BS to BS), 2) to store the data requested by the user in case of disconnection of the monitor, and 3) to refresh the data presented to the user in an efficient manner. To achieve the third task, the MonitorTracker agent creates a *Tracker agent* on the BS of each referenced object.
- *Tracker agent*. It is a mobile agent created by the MonitorTracker to track a concrete referenced object. It performs three main tasks: 1) to keep close to its referenced object, 2) to detect and process the new location of its referenced object, and 3) to detect and process the new locations of target objects corresponding to its tracked referenced object. For the third task, the Tracker agent creates an *Updater agent* on each BS whose area intersects the *extended area* of its referenced object. Section 4.1 explains how extended areas are obtained from the user query.
- *Updater agent*. It is a static agent whose goal is to detect the location of target objects under the coverage of its BS and communicate the necessary data to its Tracker agent.

The location of objects is obtained by querying a database residing on each BS. We assume that BSs store three attributes for every object: id (the object identifier), x and y (the absolute coordinates of the object).

3. Query Language

In this section we provide a brief description of the kind of queries that our system manages. We show here the proposed SQL-like syntax for location-dependent queries:

SELECT	*projections*
FROM	*set-of-objects*
WHERE	*boolean-conditions*

where *projections* is the list of attributes that we want to retrieve from the selected objects, *set-of-objects* is a list of object classes or *location-dependent constraints* (see later) that identify the objects interesting for the query (aliases can be used), and *boolean-conditions* is a boolean expression that selects objects from those included in *set-of-objects* by restricting their attribute values.

The main location-dependent constraints defined in our system are:

- *Distance(ref-obj, target-obj)*. It obtains the distance[3] between *ref-obj* and *target-obj*.
- *Nearest_n(ref-obj, target-class)*. It returns the *n* instances of *target-class* that are the nearest to *ref-obj*.
- *Furthest_n(ref-obj, target-class)*. It is the opposite constraint to "nearest_n".
- *Inside_r(ref-obj, target-class)*. It returns the instances of *target-class* that are inside a circular area of radius *r* around *ref-obj*.

[3] In our prototype, distance is measured in miles. We plan to extend the query language to deal with several measure units.

- *Outside_r(ref-obj, target-class)*. The complementary constraint of "inside_r".

Due to space limitations we cannot include in this paper details about the processing of each location-dependent constraint.

4. Query Processing Approach

We use the sample query, presented in Section 1, to introduce the query processing steps. Let us suppose that the user poses the sample query to the Query Processor with the help of a GUI. In this section we detail how the query is executed and updated continuously. In Figure 1 we present a possible scenario for the example.

Four main tasks are performed in the query processing approach: 1) *Analysis of the user query*, 2) *Obtaining of DB queries*, 3) *Initialization of DB queries*, and 4) *Refreshment of results*. We explain these steps in the following.

4.1. Analysis of the user query

The Query Processor transforms the user query into an intermediate specification. Concretely it obtains: (1) for each constraint in the query, the *referenced object and its target classes*, and (2) for each target class of each referenced object, a *relevant area*. Furthermore, by considering the maximum speed of relevant objects, the semantics of the constraint, and the answer refreshment frequency, an *extended area* for each relevant area is obtained. In the sample query, and using the estimation of our prototype[4], we obtain the following areas[5]:

referenced object	relevant area	extended area
car38	7 miles	7.6 miles
policeCar5	5 miles	5.9 miles

Extended areas are used by the Query Processor to build *DB queries* that constraint the target objects in which we are interested in (in the example, for the first constraint, those police units whose location is inside 7.6 miles around 'car38'). However, only target objects located inside the relevant area will be shown to the user. Target objects inside the extended area but outside the relevant area are considered by the Query Processor as candidates to enter the relevant area in the gap between refreshments of data shown to the user. Thus, by dealing with extended areas, the Query Processor avoids very frequent requests about relevant objects locations.

[4]We consider that the maximum speed of cars is 100 mph, the maximum speed of (the fastest) police unit is 120 mph, and the frequency of refreshment is 10 seconds.

[5]In the example, each referenced object ('car38' and 'policeCar5') has one target class only (police units and police cars, respectively).

4.2. Obtaining of DB queries

The goal is to transform a location-dependent query into queries over tables that store information about mobile objects: for each target class of each referenced object, one SQL[6] query is obtained. We present here the resulting DB queries in the example:

referenced object: car38, *target class*: policeUnit
SELECT id, x, y FROM policeUnit WHERE sqrt((x-ref$_x$)^2+(y-ref$_y$)^2))<=7.6 and not(busy)

referenced object: policeCar5, *target class:* policeCar
SELECT id, x, y FROM policeCar WHERE sqrt((x-ref$_x$)^2+(y-ref$_y$)^2))<=5.9 and id<>'policeCar5'

Notice that the above queries are based on the location of referenced objects. For the first query, ref_x and ref_y must correspond to the location of 'car38', and for the second query, ref_x and ref_y must correspond to the location of 'policeCar5'.

4.3. Initialization of DB queries

The DB queries must be executed on the BSs of referenced objects. For this task, the Query Processor must obtain which BS provides coverage to each referenced object[7]. The following are the main tasks performed in this step:

1. The Query Processor sends a MonitorTracker agent to the BS that provides coverage to the monitor. The goal of this agent is to minimize the wireless communications with the monitor. The MonitorTracker agent will follow the monitor when it moves to another BS. In the example, the MonitorTracker is sent to BS6 (see Figure 1).

2. The MonitorTracker sends a Tracker agent to the BS of each referenced object, initialized with the DB queries that it must execute there. In the example, the MonitorTracker sends one Tracker agent to BS4 (to track 'car38') and another Tracker agent to BS1 (to track 'policeCar5'). In the example, the Tracker agent on BS4 will be initialized with the first SQL query, and the Tracker agent on BS1 will be initialized with the second SQL query.

3. Each Tracker agent obtains, for each DB query, the BSs whose coverage area intersects the extended area

[6]We assume that object attributes (location and other features) are stored in a relational database on BSs but any other data organization like plain files could be used.

[7]We suppose that there exists a mechanism provided by the mobile network infrastructure that, given the id of a mobile object, returns the BS that provides coverage to such a mobile object.

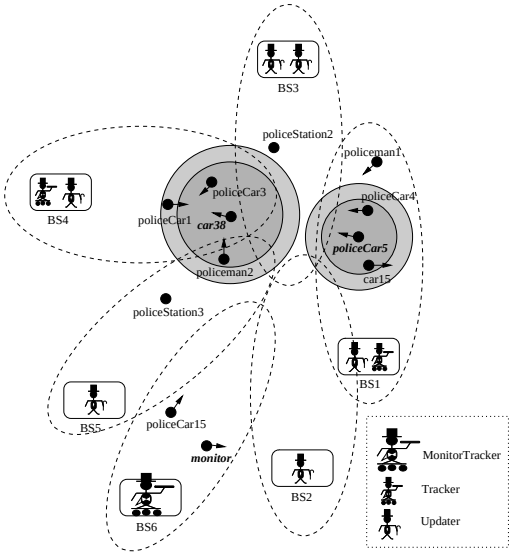

Figure 1. Scenario for the sample query

of such a DB query. The Tracker agent creates one Updater agent on each of those BSs. The idea is to keep an eye on any possible way to enter the extended area of a referenced object. Updater agents are initialized with the corresponding DB query after the Tracker agent replaces ref_x and ref_y by the current location of its referenced object. In the example, the Tracker agent on BS4 sends Updater agents to BS3, BS4, and BS5. Those Updater agents are initialized with the first SQL query after replacing ref_x and ref_y by the current location of 'car38'. Moreover, the Tracker agent on BS1 sends Updater agents to BS1, BS2, and BS3, and initializes them with the second SQL query after replacing ref_x and ref_y by the current location of 'policeCar5'.

4. Each Updater agent executes its DB queries against the database of the BS where it resides. These data are sent to the corresponding Tracker which processes the different answers and transmits the results to the MonitorTracker. The MonitorTracker combines the results coming from the different Trackers and sends the final answer to the monitor. In the example, the data returned to the Query Processor (the relevant objects inside extended areas) are the following:

Constraint-1	Constraint-2
policeCar3	policeCar4
policeCar1	
policeman2	

However data presented to the user are different from the above because 'policeCar1' is outside the relevant area of 'car38' (see Figure 1), therefore it is not shown.

Notice that the only wireless data transfer occurs when the Query Processor sends the MonitorTracker to the BS of the monitor and when the Query Processor obtains the final answer from the MonitorTracker. Any other communication occurs among BSs using a fixed network.

4.4. Refreshment of results

Data obtained by the Query Processor about target objects become obsolete whenever those target objects move to another location. Thus, Updater agents must execute their DB queries with a certain frequency (we call it *refreshment frequency*). When a new location of a target object is detected (or when a new target object becomes relevant) these data will be sent to the corresponding Tracker agent, MonitorTracker and monitor, if necessary. In this way, data presented to the user are "always" up to date.

Furthermore, queries executed by Updater agents could become obsolete whenever the corresponding referenced object moves. Thus, Tracker agents must obtain the referenced object location with a certain frequency (we call it *tracking frequency*). When a new relevant location is obtained, the new ref_x and ref_y are used to construct the new DB queries which are sent to the corresponding Updater agents. Moreover, as the referenced object has moved, and therefore the relevant and extended areas around it have moved too, the Tracker agent will check the new BSs whose coverage area intersects the new extended area. The Tracker agent could send a kill signal to its unnecessary Updater agents and create new Updater agents, if needed. The new location of the referenced object will be sent to the monitor (through the MonitorTracker agent).

Moreover, if a referenced object disappears from its current BS, its Tracker agent will ask for it to its neighbor BSs. When the new BS that provides coverage to the referenced object is obtained, the Tracker agent moves there and recalculates the new BSs whose coverage area intersects the extended area of the referenced object.

5. Related Work

The following are some approaches related to our proposal. See [9] for complementary works that propose several solutions to the problem of locating objects in a mobile environment.

The closest work to our project, as far as we know, is the DOMINO project [14, 13] which proposes a model to represent moving objects in a database. They use FTL (Future Temporal Logic) [11] as query language. FTL augments SQL with temporal and spatial operators. However we decided not to deal with temporal operators because they are only useful for environments where paths of moving objects

are known *a priori*. We are thinking of contexts where objects can move independently and that is the reason for dealing with a good set of location-aware constraints, including a 'nearest' constraint. Moreover, as opposite to DOMINO's approach, we advocate a completely decentralized solution for storing data about moving objects on BSs, which we consider more suitable for a mobile environment.

In the DATAMAN project [4], they present location-dependent queries as a challenge problem but, as far as we know, they do not propose any solution to that problem.

The Rome architecture [3] manages location and time-dependent triggers. However they do not provide a completely decentralized architecture (mobile agent technology may be useful for their approach).

Other complementary works are those that deal with location-dependent web pages [1] and location-dependent data queries on the network layer [5]. However, these works do not deal with some problems that we manage, like continuous queries.

6. Conclusions and Future Work

In this paper we have presented a system that deals with continuous queries and which is based on mobile agents to track objects relevant to a location-dependent query. We have introduced the query processing approach and a brief summary of the kind of queries that our system can process. The main features of our proposal are:

- We advocate a solution based on mobile agents to track objects involved in a query. So, the query processor is relieved from such a duty and the communication among modules is reduced. We also consider taking advantage of fixed networks whenever possible.
- Data presented to the monitor are automatically updated by the system, by tracking and recalculating the corresponding queries in an efficient manner.
- We have defined an SQL-like query language to build different queries concerning location and other features of objects in the scenario.
- The proposed solution is completely decentralized and therefore it is highly extensible and scalable.

Currently, we are working on the following open issues: 1) we plan to analyze analytically (using Petri nets) the behavior of the system in order to decide when we should reuse existing agents (to avoid, for example, having two Updater agents on the same BS); and 2) agents could approximate the location of relevant objects using extrapolation techniques on objects' paths, to minimize the communication among agents when relevant objects follow predictable paths.

References

[1] A. Acharya, T. Imielinski, and B. Badrinath. Dataman project: Towards a mosaic-like location-dependant information service for mobile clients. *MOBIDATA*, 1(2), April 1995.

[2] D. Barbará. Mobile computing and databases - a survey. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):108–117, Jan-Feb 1999.

[3] A. C. Huang, B. C. Ling, S. Ponnekanti, and A. Fox. Pervasive computing: What is it good for? In *Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access, August 20, 1999, Seattle, WA, USA*, pages 84–91. ACM, 1999.

[4] T. Imielinski. Mobile computing: Dataman project perspective. *Mobile Networks and Applications (MONET)*, 1(4):359–369, 1996.

[5] T. Imielinski and A. Acharya. The network as a database machine. *MOBIDATA*, 1(2), April 1995.

[6] T. Imielinski and B. Badrinath. Mobile wireless computing: Challenges in data management. *Communications of the ACM*, 37(10):18–28, October 1994.

[7] T. Imielinski and H. Korth, editors. *Mobile Computing*. Kluwer Academic Publishers, 1996.

[8] E. Pitoura and G. Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publishers, 1998.

[9] E. Pitoura and G. Samaras. Locating objects in mobile computing. *IEEE Transactions on Knowledge and Data Engineering*, 2001. To appear.

[10] K. Porkaew. *Database Support for Similarity Retrieval and Querying Mobile Objects*. PhD thesis, University of Illinois at Urbana-Champaign, 2000.

[11] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proceedings of the Thirteenth International Conference on Data Engineering (ICDE'97), Birmingham U.K., IEEE Computer Society , ISBN 0-8186-7807-0*, pages 422–432, April 1997.

[12] Y. Villate, D. Gil, A. Goñi, and A. Illarramendi. Mobile agents for providing mobile computers with data services. In *Proceedings of the Ninth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 98)*, 1998.

[13] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *invited paper, special issue of the Distributed and Parallel Databases Journal on Mobile Data Management and Applications*, 7(3):257–287, 1999.

[14] O. Wolfson, A. P. 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.