Location-Dependent Queries in Mobile Contexts: Distributed Processing Using Mobile Agents

Sergio Ilarri, Eduardo Mena, and Arantza Illarramendi

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. In this paper, we present a system that supports distributed processing of continuous location-dependent queries in mobile environments. The system that we propose presents the following main advantages: 1) it is a general solution for the processing of location-dependent queries in scenarios where not only the users issuing queries, but also other interesting objects can move; 2) it performs an efficient processing of these queries in a continuous way; 3) it is especially well adapted to environments where location data are distributed in a network and processing tasks can be performed in parallel, allowing a high scalability; and 4) it optimizes wireless communications. We use mobile agents to carry the processing tasks wherever they are needed. Thus, agents are in charge of tracking the location of interesting moving objects and refreshing the answer to a query efficiently. We evaluate the usefulness of the presented proposal showing that the system achieves a good precision and scales up well.

Index Terms—Distributed processing of continuous location queries, tracking moving objects, mobile agents.

1 Introduction

The continuous development and improvement of wireless networks and mobile computing devices, together with their challenging limitations, has motivated an intense research effort in mobile data services; according to Strategis Group, there will be one billion wireless subscribers worldwide on 3G networks by the year 2010.[1] While most of these services are the counterpart of those available in desktop computers, there exist other applications that exploit the dynamic features of the mobile environment to provide the user with context-aware information. Specifically, perspectives are that the location-based services economy will reach $100 billion by 2008 [34]. In this paper, we focus on continuous location-dependent queries, which are still a subject of research mainly due to the lack of a general architecture that is well adapted to process them efficiently. 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. In the following, we introduce the features considered by our approach, first for the location-dependent queries and next for the query processing. We finish this section with a summary of the contributions of our proposal.

1. http://www.mobileinfo.com/Market/market_outlook.htm (accessed 30 May 2005).

- *S. Ilarri and E. Mena are with the Department of Computer Science and System Engineering, Centro Politécnico Superior, University of Zaragoza, María de Luna 3, 50018 Zaragoza, Spain. E-mail: silarri@unizar.es, emena@unizar.es.*
- *A. Illarramendi is with the Department of Computer Languages and Systems, University of the Basque Country, Apdo. 649, 20080 Donostia/San Sebastián, Spain. E-mail: jipileca@si.ehu.es.*

Manuscript received 20 Oct. 2004; revised 1 Apr. 2005; accepted 6 May 2005; published online 15 June 2006.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0288-1004.

1.1 Location Queries: Supported Features

In the following, we present the three main features of the location queries processed by our system:

- The queries are treated as *continuous queries* since the set of objects that satisfy the conditions specified in a location-dependent query can change very quickly due to the mobility of the involved objects (its answer depends on the location of such moving objects). For example, if a user asks about the nearest hotel while driving through the downtown of a large city, the expiry time [35] of the returned answer could be very short because with small displacements of the car a different hotel could be the nearest. Thus, the answer to continuous queries must be updated continuously at a certain *refreshment frequency* (e.g., update the answer every 5 seconds) until they are cancelled by the user, as opposite to the traditional instantaneous queries for which only a single answer is obtained. Moreover, we adopt a different approach from those who propose updating the answer as soon as a change is detected; the cost of those approaches is unbounded as the answer could be changing continually!
- The answer to a location query can depend on the location of any object in the scenario. This kind of location queries are called *moving queries* in the literature [6] because the geographic areas they are interested in change with the movements of the objects referenced by such queries. By contrast, most commercial solutions only allow queries about static

1536-1233/06/$20.00 © 2006 IEEE Published by the IEEE CS, CASS, ComSoc, IES, & SPS

objects (e.g., restaurants, gas stations) around the current location of the user that posed such queries.
- The user could be interested not only in the set of objects that satisfy the constraints in the query (as they do, for example, in [6], [24]), but also in their current geographic location (for instance, to locate such objects on a map). Therefore, the query processing approach must aim at providing *location-aware answers*, which also advises the refreshment of the answer with a certain frequency.

1.2 Location Query Processing: Challenges and How to Face Them

Location query processing techniques should meet the following two features: 1) to be suitable for a distributed environment and 2) to refresh the answer to continuous queries efficiently. We explain these goals in the following.

1.2.1 Distributed Query Processing

While the inherent distribution of the moving objects themselves suggests a distributed approach for the query processing, many existing works reduce the problem to a centralized query processing [28], [41], [42]. In our opinion, for scalability, performance, and feasibility reasons, it is not convenient to assume that there exists a centralized computer which is aware of the locations of *all* the moving objects of interest in the scenario (there could be millions of moving objects spread out over a very large geographic area). While other works such as MobiEyes [6] distribute the query processing, they suffer several disadvantages. For example, their approach can overload user devices and increase their wireless communications efforts, as they rely on certain availability of processing and memory capabilities of the user devices in order to process queries on them.

As opposed to these proposals, we advocate a distributed approach where a set of fixed computers, called *proxies*, manage the location information of moving objects within a certain geographic area, which we call *proxy area*. For example, in a cellular network [26], a proxy area is obtained as the union of the coverage areas of one or more base stations. At each proxy, a *Data Management System* (DMS) module is in charge of storing and providing location data about moving objects within that proxy area. We would like to stress that we are not concerned about the problem of how a DMS is aware of the location of moving objects; for example, it could receive location updates from GPS-equipped moving objects (that data can be efficiently stored in a DOMINO database [41] or processed on the fly [2]), or it could explicitly locate the moving objects when needed.

Thus, a query processing approach distributed among proxies provides important added benefits:

- *User devices are not overloaded and wireless communications with them are not increased* as the query processing is performed on wired computers.
- *The performance of the query processing is increased.* Thus, queries concerning different geographic areas (e.g., queries about taxi cabs around the O'Hare and Midway airports in Chicago) are processed in parallel as they do not interfere each other.
- *The addition of new proxies* and/or redefinition of their coverage areas to support more users/objects and/or balance the system load *can be done transparently.*
- *The service availability can be easily enhanced,* as the replication of data and functionalities is also easy to achieve with a distributed architecture.

1.2.2 Efficient Refreshment of the Answer

It is not possible to provide a completely precise snapshot of the locations of moving objects, due to two reasons: 1) the locations of all the moving objects cannot be measured and obtained at exactly the same time instant and 2) any existing positioning method incurs some error. However, by considering a query whose answer is continuously updated, we get a snapshot of the scenario good enough to show how it evolves over time.

Special difficulties arise from the need of keeping the answer to continuous queries up to date while optimizing the wireless communications. If we consider a continuous query as just a sequence of instantaneous queries that are periodically evaluated, we will probably not be able to refresh the answer with the required frequency, due to the involved remote data access delays and initialization overhead. To process continuous queries efficiently, we propose a dynamic tracking architecture that adapts itself to the movements of the relevant objects. For that task, we advocate the use of mobile agents to track the interesting moving objects and optimize the wireless communications. Mobile agents can move to remote computers to achieve their goals, avoiding the need of installing specialized server processes on every machine to provide remote access. Besides, mobile agents ease the addition of new services once the system is working: New agents with new functionalities can travel to the user device or proxies without reinstalling anything there. Mobile agents are very convenient for this context[2] not only from a design and implementation point of view (they "implement" a distributed query processing approach easily), but they also exhibit a good performance [23], [33].

1.3 Summary of Our Proposal and Main Contributions

We present in this article a flexible, extensible, and scalable distributed architecture based on mobile agents for the continuous processing of location-dependent queries. As far as we know, *no other work proposes a completely decentralized solution for continuous moving queries without overloading user wireless devices.* Furthermore, as opposed to other proposals, we aim for providing *location-aware answers* instead of just computing the set of objects that satisfy certain conditions. The use of mobile agents brings many benefits in this context. Last but not least, our solution does not make any assumption regarding: 1) The precision of the location data, that can be obtained using different positioning methods with different precision [9], whether they are GPS-based or network-based, 2) the way that this location information is managed (e.g., stored in databases, estimated using predefined trajectories, or pulled on demand from the

2. The study of the security implications is out of the scope of this paper. We refer interested readers to [39].

```
SELECT   policeUnit.id, policeCar.id
FROM     policeUnit, policeCar
WHERE    inside(0.56 miles, 'car38', policeUnit) AND inside(0.42 miles, 'policeCar5', policeCar)
         AND policeCar.id <> 'policeCar5'
```

Fig. 1. Sample query.

moving objects themselves), and 3) the set of proxies that manage and provide these data. Our system works for any number of data sources and even when these sources, and the way that the data are distributed among them, change dynamically.

In our proposal, the information presented to the user is as recent as possible, which makes our system suitable for both online (see [40] for an example) and offline analysis (see [43]) of location data. However, in this work, we focus on online query processing, as it is the most demanding scenario.

In the rest of the paper, we describe and validate our query processing approach, whose basic architecture was presented in [10]. In Section 2, we describe the types of location-dependent queries that we consider. We explain in Section 3 our query processing approach and how the system tracks the objects relevant to a query. In Section 4, we evaluate the performance of our approach, namely, its precision and scalability. In Section 5, we describe some related work, and finally conclusions and future work are included in Section 6.

2 Location-Dependent Queries

In this section, we present, using an SQL-like syntax, the kind of location queries that our system processes. We would like to mention that the definition of a new language is probably not necessary as we could adapt some of the existing query languages for moving objects proposed in the literature (such as [8]). In the used syntax, location queries have the following structure:

```
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 that identify the objects interesting to the query, and *Boolean-conditions* is a Boolean expression that selects objects from those included in *set-of-objects* by restricting their attribute values or demanding the satisfaction of certain *location-dependent constraints* (explained later). For example, the query in Fig. 1 asks for police units (i.e., police stations, policemen, and police cars according to the class hierarchy that we consider in our prototype) that are within 0.56 miles around car38 (a stolen car), and the police cars within 0.42 miles around policeCar5 (the current chaser police car).

For each location-dependent constraint of a query, the following definitions are managed:

- *Reference object*: the object that is the reference of the constraint. In the sample query, there exist two reference objects: car38 for the constraint *inside(0.56 miles, 'car38', policeUnit)*, and policeCar5 for the constraint *inside(0.42 miles, 'policeCar5', policeCar)*.
- *Target class*: the class of objects that are the target of the constraint. In the sample query, there exist two target classes: policeUnit for the constraint *inside(0.56 miles, 'car38', policeUnit)*, and policeCar for the constraint *inside(0.42 miles, 'policeCar5', policeCar)*. The instances of a target class are called *target objects*.
- *Relevant objects*: the reference and target objects involved in the constraint. In the first constraint of the sample query, the relevant objects are car38 and the police units, while in the second constraint they are policeCar5 and the rest of police cars.

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

- *Inside(r, ref-obj, target-class)*. It filters out the instances of *target-class* that are not inside a circular area of radius *r* around *ref-obj*, returning true otherwise.
- *Outside(r, ref-obj, target-class)*. The complementary constraint of "inside."
- *Nearest(n, ref-obj, target-class)*. It filters out the instances of *target-class* that are not among the *n* nearest to *ref-obj*, returning true otherwise.
- *Furthest(n, ref-obj, target-class)*. It filters out the instances of *target-class* that are not among the *n* furthest to *ref-obj*, returning true otherwise.

Besides, we define the function *Distance(ref-obj, target-obj)*, which obtains the distance between *ref-obj* and *target-obj*, and can be used in the FROM clause.

As shown in Table 1, most of these constraints can be expressed in terms of an *inside* constraint, so in the rest of the paper we will focus on the processing of *inside* constraints. In the table, we assume that there is a maximum radius r' that delimits the area of interesting objects[3] and ROWNUM $\leq n$ is an expression (borrowed from Oracle SQL syntax) that delimits the size of the output. Of course, some optimizations are possible during the processing of these constraints. For example, an *outside* constraint does not need to be implemented as two *inside* constraints: instead, the system can retrieve the objects in the difference set with just one query. Constraints such as *nearest* [30] have been extensively studied in a centralized context, although these works focus on returning the objects in the answer, not also their current locations as we do.

3. In practice, it makes no sense to query about all the objects in the world, so r' determines the maximum geographic area of interest.