An Approach to Process Continuous Location-Dependent Queries on Moving Objects with Support for Location Granules

Sergio Ilarri*a, Carlos Bobeda, Eduardo Menaa

aIIS Department, University of Zaragoza, Maria de Luna 1, 50018, Zaragoza, Spain

Abstract

Location-based services have attracted the attention of important research in the field of mobile computing. Specifically, different mechanisms have been proposed to process location-dependent queries. In the above mentioned context, it is usually assumed that the location data are expressed at a fine geographic precision. However, a different granularity may be more appropriate in certain situations. Thus, a location resolution higher than required may even be inconvenient or not understandable by the user (for example, if the user expects a city name as an answer and instead the system provides the latitude/longitude coordinates). Moreover, if the locations presented to the user need to be refreshed automatically as the objects move, it is obvious that maintaining up-to-date GPS-like geographic coordinates would be more expensive in terms of processing and communication. Unfortunately, the existing approaches assume queries whose locations are always given with maximum precision (i.e., GPS locations).

In this paper, a distributed query processing approach that adapts itself to the level of the location resolution required is presented. Thus, it supports continuous location-dependent queries based on the required terminology for the locations, depending on the granularity used (e.g., GPS, cities, states, provinces, or any other predefined geographic area). For the above mentioned purpose, location granules can be defined to specify the semantics appropriate for the queries and/or the way the results should be presented. A prototype showing the functionality and benefits of the approach has been implemented and used in an extensive experimental evaluation. The proposal not only increases the flexibility and expressive power of the queries considerably but also performs efficiently.

Keywords:
Location-based services, location granules, location-dependent constraints, continuous and distributed query processing
1. Introduction

Recently, in the mobile computing field, there has been an intensive research effort in location-based services (Schiller and Voisard, 2004), which provide value-added data by considering the locations of the mobile users to offer more customized information. One of the greatest challenges in location-based services is the efficient processing of continuous location-dependent queries (e.g., tracking cars near a moving user). Thus, these queries require a continuous monitoring of the locations of the relevant moving objects to efficiently keep the answers up-to-date. For example, a user with a PDA may want to locate the available taxi cabs that are near him/her while he/she is walking home on a rainy day (ordering a taxi is an example of a location-dependent service considered in Veijalainen and Weske (2003)). The answer to this query must be continuously refreshed, as it can undergo immediate changes due to the movements of the user and the taxi cabs. Moreover, even if the set of taxis satisfying the query condition does not change, their locations and distances to the user could change continuously. Therefore, the answer to the query must be updated with the new location data (e.g., to update the locations of the taxis on a map).

The existing work on location-dependent query processing implicitly assumes GPS locations for the objects in a scenario (e.g., Sistla et al. (1997); Prabhakar et al. (2002); Mokbel et al. (2005); Cai et al. (2006); Gedik and Liu (2006); Ding et al. (2008); Ilarri et al. (2010)). However, some applications do not require location data at GPS resolution, and a coarser representation may be more appropriate for them. For example, a train tracking application would need to just consider in which city a train is currently in, and not its exact coordinates. For such applications, it is useful to define the concept of location granule as a set of physical locations. In the previous example, every city would correspond with a location granule. The idea is that it should be possible to express the queries and retrieve the results according to the concept of “location” required, whether in terms of GPS locations (the finest type of location granule possible) or locations at a different resolution (e.g., freeways, buildings, offices in a building, etc.). The use of location granules can have an impact on:

- **The presentation of results.** The user may want to retrieve the precise geographic coordinates of the objects in the answer to the query. Alternatively, he/she may prefer a different location granularity (e.g., the cities where the interesting objects are) as it is more appropriate for his/her context. The answer can be presented using different mechanisms (e.g., different types of graphical, sound-based, or textual representations), which are independent of the required location granularity.

- **The semantics of the queries.** It is possible to easily define the queries using a specific location terminology (i.e., based on location granules). In this case, the answer to the query depends on the interpretation of the location granules. For example, the user may be interested in the cars that are near the city where another car is currently present. In the above mentioned example, the user may have no idea about the geographic coordinates or the boundaries of the cities. Thus, the management of the granules must be performed transparently.
The performance of the query processing. Some tasks in the continuous query processing demand less resources when coarse location granules are used. Thus, the objects would move less frequently between granules, and keeping track of their current locations is easier than if precise GPS locations must be considered.

In this paper, the utility of dealing with locations expressed at different levels of granularity when processing location-dependent queries is emphasized. Besides, a suitable query processing approach is presented. It should be noted that the proposal could benefit from the support offered by the existing spatial database management systems. However, this work focuses on the problem of distributed processing of continuous location-dependent queries with location granules about moving objects in mobile environments, which has not been studied before in the literature. In these environments two main elements can be identified: moving objects (each one attached to a mobile device that allows its detection) and fixed computers (called proxies in Ilarri et al. (2006a)) that can detect the moving objects within a certain geographic area. In this context, new opportunities and challenges arise. Specifically, by processing the queries in a distributed way on proxies that manage data and queries concerning different geographic areas, several benefits are obtained, such as:

1. The mobile devices of the users are not overloaded with query processing and wireless communication tasks.
2. The performance and scalability of the query processing is increased, as queries concerning different geographic areas are executed in parallel without interferences between them.
3. The addition of new proxies and/or the redefinition of their coverage areas to support more users/objects and/or balance the system load can be done transparently.
4. The service availability can be easily enhanced (by replicating the data and functionalities).

However, processing location-dependent queries in such a distributed environment is not an easy task. In this sense, the following challenges could be highlighted, that are overcome by the system described in this paper:

1. Keeping the answer to the continuous queries up to date while optimizing the wireless communications and the query processing efforts.
2. Keeping track of the proxies that store the relevant data.
3. Obtaining consistent snapshots of the environment (i.e., data from the different proxies involved should be obtained at approximately the same time instant).
4. Managing the location granules efficiently to allow a scalable continuous query processing about moving objects.
5. Analyzing the advantages that location granules can provide in this context.

---

1This paper extends and improves considerably the initial proposal of the use of location granules in location-dependent queries presented in Ilarri et al. (2007), including the integration with a distributed location-dependent query processing system and an extensive experimental evaluation.
The structure of the rest of the paper is as follows. In Section 2, the importance of location granules and how they can be used to express queries with semantics that are useful to the user is justified. In Section 3, the architecture proposed to manage location granules is described. In Section 4, the approach proposed to manage location granules in location-dependent constraints is presented. In Section 5, the integration of the proposal with an existing system that processes continuous location-dependent queries in distributed environments is described. In Section 6, some aspects of the prototype implemented to test the proposal are explained, and the first extensive experimental evaluation that proves the feasibility of processing location-dependent queries with location granules in mobile environments is presented. In Section 7, some related work is described. Finally, some conclusions and possible extensions of the current work appear in Section 8. Moreover, Appendix A describes the query processing algorithm in detail. Two additional electronic appendices (hosted at http://webdiis.unizar.es/~silarri/PUBLICATIONS/jss2011/) contain an analysis of the complexity of this algorithm as well as some supplementary experiments.

2. Location-dependent queries with location granules

A location granule is composed of one or more geographic areas which identify a set of GPS locations under a common name (which is similar to the concept of place in Hightower (2003); Hoareau and Satoh (2007, 2009) or spatial granule in Belussi et al. (2009)). For example, Madrid is a location granule of type city, such that it can be said that a certain car is in Madrid or in the location defined by the coordinates (x,y), depending on the location granularity required (city or GPS granularity, respectively). The focus in this paper is on the query processing of location-dependent queries with location granules. It considers that a location-dependent query is any query whose answer depends on the locations of certain objects (the mobile user and/or other interesting objects). For example, a query that retrieves the locations of the taxis around a person who is searching for a cab is an example of a location-dependent query. In comparison with other proposals, such as Seydim et al. (2001), the above mentioned definition of location-dependent query is quite general (Ilarri et al., 2010). Besides, the possibility to use location granules also contributes to extending the range of location-dependent queries that can be expressed. For example, a continuous query such as “retrieve the cars that are within 100 miles of the city where car38 is, showing their locations with city granularity” (i.e., indicating the city where each retrieved car is) could be submitted to keep track of the interesting moving objects and their current cities.

An SQL-like syntax will be used to express the queries. Although an existing spatial query language could be adopted, such as SQL/MM (Stolze, 2003) or Spatial SQL (Egenhofer, 1994), the above mentioned syntax allows for emphasizing the use of location granules and stating the queries concisely. For example, the first query in Figure 7 (page 15) could be expressed in SQL/MM as follows: “SELECT Car.id FROM Car, MapProvinces map WHERE ST_CONTAINS(ST_BUFFER(ST_UNION(ARRAY(SELECT granule FROM MapProvinces, Car WHERE Car.id = “car38” AND ST_CONTAINS(granule, Car.position))), 130 miles, 32), Car.position)”. This is quite verbose for the purpose of this work. Similarly, OQL (Object Query Language, pro-
posed by the Object Data Management Group (ODMG) could be adopted, which is described in Cattell et al. (2000). With OQL, the previous query could be expressed as follows (assuming the existence of a global object SP that is an instance of SpatialProcessor, a class that acts as an interface encapsulating the spatial operations proposed in the OpenGIS Specification): “SELECT c.id FROM c in Cars WHERE SP.inside(c, Distance(130, “miles”), element(SELECT ref FROM ref in Car WHERE ref.id = “car38”)). Although the use of OQL is possible and would help in having a more “standard” language, for the purpose of clarity a different syntax, which highlights the use of constraints with location granules in a simple way, is used in this paper. New standardization efforts for object databases are currently underway (Card, 2007).

The general structure of location-dependent queries, as proposed in Ilarri et al. (2006a), is as follows:

```
SELECT projections FROM sets-of-objects WHERE boolean-conditions
```

where sets-of-objects is a list of object classes that identify the kind of objects which are interesting for the query, boolean-conditions is a boolean expression that selects objects from those included in sets-of-objects by restricting their attribute values and/or demanding the satisfaction of certain location-dependent constraints, and projections is the list of attributes or location granules that must be retrieved from the selected objects. The queries are interpreted as continuous queries (Terry et al., 1992), whose answer must be refreshed automatically by the system until the user cancels the query.

The detailed syntax of the types of queries considered in this work is shown in Figure 1, where it can be seen how location granules can be used in the queries. Non-terminals in the grammar are represented with initial uppercase and terminals are in lowercase (keywords) or surrounded by single quotes (literals). The start symbol of the grammar is Query. The symbol gr stands for granule: gr(map-id, obj-id) indicates that the location of the object named obj-id must be interpreted as a granule in the location granule map identified by map-id (a location granule map is a set of granules, as described in Section 3.1), and similarly gr(map-id, class) generalizes this to all the objects of class. It should be noted that there is a difference between a location appearing in the SELECT clause (identified with Loc-Select) and a location appearing as part of a location-dependent constraint (Loc-Ref or Loc-Target). In the first case the second argument of gr must be a class name, whereas in the second case it can be a class name (for Loc-Target) or an object identifier (for Loc-Ref). This difference is consistent with the way projections are performed in standard SQL. For example, gr(“city”, Car) in a SELECT clause would imply retrieving the granule (of the location granule map named “city”) for each object of class Car retrieved by the query. The non-terminal Object-id represents a value of the attribute id of the datatype Object (defined in Section 3.1). Thus, it is not an object identifier (OID) in the object-oriented sense;

---

2Moving objects have a unique identifier and are classified according to a class hierarchy. For example, the class policeUnit includes the classes policeman, policeCar and policeStation.
the same can be said about the non-terminal *Map-id* regarding the attribute *id* of the datatype *Location Granule Map*.

**General query structure**

<table>
<thead>
<tr>
<th>Production</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Query</strong></td>
<td>→ <em>select</em> Projections from Class-names <em>(where Conds)</em>?</td>
</tr>
<tr>
<td><strong>Class-names</strong></td>
<td>→ Class-name <em>(,’ Class-name)</em></td>
</tr>
<tr>
<td><strong>Projections</strong></td>
<td>→ Attr-Loc-Select <em>(,’ Attr-Loc-Select)</em></td>
</tr>
<tr>
<td><strong>Attr-Loc-Select</strong></td>
<td>→ attribute</td>
</tr>
<tr>
<td><strong>attribute</strong></td>
<td>→ Qualified-attrib</td>
</tr>
<tr>
<td><strong>Qualified-attrib</strong></td>
<td>→ Class-name ‘.’ Unqualified-attrib</td>
</tr>
<tr>
<td><strong>Loc-Select</strong></td>
<td>→ Object-id ‘.’ ‘loc’</td>
</tr>
</tbody>
</table>

**Conditions can be standard conditions on attributes or location-dependent conditions**

<table>
<thead>
<tr>
<th>Production</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Conds</strong></td>
<td>→ Cond *((and</td>
</tr>
<tr>
<td><strong>Cond</strong></td>
<td>→ (Bool-Cond</td>
</tr>
<tr>
<td><strong>Bool-Cond</strong></td>
<td>→ attribute Comp Value</td>
</tr>
</tbody>
</table>

**Location-dependent conditions**

<table>
<thead>
<tr>
<th>Production</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LDQ-Cond</strong></td>
<td>→ <em>inside</em> <em>(’ Args-Inside ‘)</em></td>
</tr>
<tr>
<td><strong>Arg-Inside</strong></td>
<td>→ Radius ‘.’ Loc-Ref ‘.’ Loc-Target</td>
</tr>
<tr>
<td><strong>Loc-Ref</strong></td>
<td>→ Object-id</td>
</tr>
<tr>
<td><strong>Loc-Target</strong></td>
<td>→ Class-name</td>
</tr>
<tr>
<td><strong>Radius</strong></td>
<td>→ Real Units</td>
</tr>
</tbody>
</table>

**Basic grammar productions**

<table>
<thead>
<tr>
<th>Production</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>String</strong></td>
<td>→ ([a-z]</td>
</tr>
<tr>
<td><strong>Real</strong></td>
<td>→ ([0-9]+) *(.’ [0-9]+)#</td>
</tr>
<tr>
<td><strong>Class-name</strong></td>
<td>→ String /* Name of a class of objects */</td>
</tr>
<tr>
<td><strong>Unqualified-attrib</strong></td>
<td>→ String /* Name of an attribute for a selected class */</td>
</tr>
<tr>
<td><strong>Object-id</strong></td>
<td>→ “ String ” /* Identifier of an object */</td>
</tr>
<tr>
<td><strong>Map-id</strong></td>
<td>→ “ String ” /* Identifier of a granule map */</td>
</tr>
<tr>
<td><strong>Gr-id</strong></td>
<td>→ “ String ” /* Identifier of a granule */</td>
</tr>
<tr>
<td><strong>GPS-coord</strong></td>
<td>→ <em>(’ Real ‘,’ Real ‘)</em> /* Two dimensions are assumed */</td>
</tr>
<tr>
<td><strong>Units</strong></td>
<td>→ meters</td>
</tr>
<tr>
<td><strong>Comp</strong></td>
<td>→ ‘=’</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>→ ([0-9]+)</td>
</tr>
</tbody>
</table>

As indicated in Figure 1, location granule maps can be referenced in the SELECT and/or in the WHERE clause of a query, depending on whether their corresponding location granules must be used for the visualization of results and/or for the processing of constraints, respectively (of course, both usages can also appear simultaneously in the same query). The above mentioned two cases are explained in the following sections.
2.1. Granule-based query projections: representation mechanisms

Location granules could be retrieved as part of the query projections. In the above mentioned case, the retrieved granules can be represented using different mechanisms, depending on the implementation of two operations used to represent a granule (showGranule and showGranuleObjects, explained later in Section 3.1).

For example, the location granules retrieved as part of the answer to a query could be shown to the user graphically. As a specific case, imagine that a user wants to track which state an interstate bus is traversing, without being interested in its exact GPS location. In such a scenario, SELECT gr(“state”, Bus) FROM Bus WHERE Bus.id=“bus25” retrieves the location granule of type state (more specifically, a granule in a granule map –i.e., set of granules– representing states) corresponding to the current location of the bus with identifier bus25. For example, the granule retrieved may be represented by highlighting the current state on a map (e.g., the black area in Figure 2.a). Alternatively, the user may want to highlight the state where the bus is located by using a bus route representation. In Figure 2.b the states are denoted by equidistant circles in a line and the current state is filled with black. It should be noted that the figure is a schematic representation of the route; for example, it does not show the real distances between the states, as this is not needed for the user’s purposes. The user may also want to use a different granularity for the locations retrieved (e.g., by using a granule map composed of granules that define cities instead of using a granule map that defines states).

The granules retrieved in the query projections can be represented in many different ways (both graphical and other types of representations). Thus, by specifying different types of location granules in the query projections, the user can obtain the locations represented as he/she needs. Furthermore, this is completely independent of the query processing (retrieval of the results). In other words, the same results retrieved can be represented using different mechanisms. Some examples of graphical and sound-based representations are available in an interactive demonstration applet.

See the second applet and/or the second demo video available at http://sid.cps.unizar.es/LOQOMOTION/loqomotion-granules.html.
2.2. Granule-based query constraints: query semantics

This section explains how it is possible to express the queries adapted to the types of locations required with the use of location granules. The possibility to reference location granule maps in query constraints enhances the expressivity of the query language. Thus, different meanings for the query are possible, depending on how the granule maps are referenced in the query constraints, and so the query processing is affected. Without loss of generality, the focus is on inside constraints, which have a general syntax as follows:

\[ \text{inside}(r, \text{obj-id}, \text{target}) \]

which retrieves the objects of a certain class \( \text{target} \) (such objects are called target objects and their class the target class) within a specific distance \( r \) (which is called the relevant radius) of a certain moving object identified by \( \text{obj-id} \) (that is called the reference object\(^4\) of the constraint). In the above mentioned general syntax, locations of the reference object and target objects are considered as GPS locations by default. However, the second and/or the third argument of the inside constraint can specify that a location has to be interpreted according to a certain granule map. Thus, \( \text{obj-id} \) can be replaced by \( \text{gr(map1-id, obj-id)} \) and \( \text{target} \) can be replaced by \( \text{gr(map2-id, target)} \), where \( \text{map1-id} \) and \( \text{map2-id} \) are the identifiers of granule maps (a granule map is a set of granules, as defined formally in Section 3.1).

If no granule map is specified, GPS locations are assumed, which could be useful, for example, for float/flight tracking. For example, the query presented in Figure 3 retrieves the cars within 130 miles of \( \text{car38} \) (a way of showing the answer to this query could be painting the cars retrieved in their current locations on a map). In this query, the reference object is \( \text{car38} \), the target class is \( \text{Car} \), and the relevant radius is 130 miles. If granule maps are associated to the inside constraint of that query, several cases (with different semantics) can be distinguished. For example, if “\( \text{car38} \)” is replaced by \( \text{gr(“province”, “car38”)} \), then the query retrieves the identifiers of the cars within 130 miles of the province where \( \text{car38} \) (the reference object of the inside constraint) is.

The management of location granules in location-dependent constraints is considered in more detail in Section 4.

```
SELECT Car.id FROM Car WHERE inside(130 miles, “car38”, Car)
```

Figure 3: Sample query not considering location granules

\(^4\)It should be noted that, according to the syntax of queries summarized in Figure 1, the second argument of inside can also be a GPS location or a fixed location granule (see the production for \( \text{Loc-Ref} \)). For the sake of simplicity, this case is omitted here; those types of spatial constraints are easier to process because they do not depend on the continuously-changing location of a reference moving object.
3. Architecture to manage location granules

First, this section introduces the definitions of some important datatypes considered in the approach adopted in this work. Next, the basic architecture considered for the processing of location-dependent queries with location granules is described.

3.1. Definition of datatypes

The three datatypes that are considered for processing location-dependent queries with location granules are: Object (O), Location Granule (G), and Location Granule Map (M). In the rest of this section, their definitions are presented by specifying their components and operations.

Definition 1. Datatype Object (O).

A value of type Object (in the following, type O) is defined by a tuple \(<\text{id}, \text{loc}, \text{class}>\), where \(\text{id}\) is the object identifier, \(\text{loc}\) is the location of the object (given by the coordinates \(\text{loc}.x\) and \(\text{loc}.y\)), and \(\text{class}\) indicates the type of object. For example, \(<"\text{car12", (x, y), \text{Car}>\) is an object of class \(\text{Car}\), named \(\text{car12}\) and located at the coordinates \((x, y)\).

There could also be other attributes specific to a certain class of objects (e.g., an object of class Vehicle may have an attribute to represent its maximum speed or its license plate number). For the purpose of simplicity, an operation \(\text{object}(\text{obj-id})\), that returns the object whose attribute \(\text{id}\) equals \(\text{obj-id}\), is also considered. Finally, a \(\text{showObject}\) operation is defined for the objects, which represents the object for a user (by using graphics, sounds, text, etc.).

Definition 2. Datatype Location Granule (G).

A value of type Granule (in the following, type G) is defined by a tuple \(<\text{id}, \text{As}>\), where \(\text{id}\) is the granule identifier and \(\text{As}\) is a set of spatial areas \(\text{As} = \{A_1,A_2,...,A_n\}\) (e.g., polygons, circles, etc.). The following operations are defined for location granules:

- \(\text{inGranule}: \text{G} \times \text{GPS} \rightarrow \text{Boolean}\), with \(\text{GPS} = \text{set of all possible GPS locations}\)

\[
\text{inGranule}(g, \text{loc}) \iff \exists A_i \in g. \text{As} \mid \text{contains}(A_i, \text{loc})
\]

\[
\text{contains}(A_i, \text{loc}) \iff \text{loc} \text{ is within } A_i
\]

For example, \(G_{\text{Chicago}} = <"\text{Chicago", \text{As}_{\text{Chicago}}>\) can represent a location granule with \(\text{id} = "\text{Chicago}\)” and its operation \(\text{inGranule}\) returns \(\text{true}\) for all the GPS locations within the boundaries of the city of Chicago (defined by \(\text{As}_{\text{Chicago}}\)). Similarly, \(G_{\text{Spain}}\) can represent a location granule with \(\text{id} = "\text{Spain}\)” and its operation \(\text{inGranule}\) returns \(\text{true}\) for all the GPS locations within the boundaries of any of the peninsular provinces of Spain or its islands. The last example emphasizes that a granule can be composed of several disconnected geographic areas.

- \(\text{inGranule}: \text{P(G)} \times \text{GPS} \rightarrow \text{Boolean}\), where \(\text{P(G)}\) is the power set of \(\text{G}\)

\[
\text{inGranule}((\{g_i\}, \text{loc}) \iff \exists g_i \in \{g_i\} \mid \text{inGranule}(g_i, \text{loc})
\]
• distCentroidGranule: \( G \times GPS \rightarrow \text{Real} \), where dist is used as a shorthand of distance

\[
dist\text{CentroidGranule}(g, loc) = \min_{A_i \in g.A} \text{dist}(loc, \text{centroid}(A_i))
\]

• distLimitsGranule: \( G \times GPS \rightarrow \text{Real} \), where dist is used as a shorthand of distance

\[
dist\text{LimitsGranule}(g, loc) = \min_{\forall A_i \in g.A} \{ \text{dist}(loc, A_i) \}
\]

\[
dist(loc, A_i) = \begin{cases} 
\min_{\forall \text{edge } l \in A_i} \{ \text{dist}(loc, l) \} & \text{if } A_i \text{ is a polygon} \\
|\text{dist}(loc, \text{center}(A_i)) - \text{radius}(A_i)| & \text{if } A_i \text{ is a circle}
\end{cases}
\]

• distGranuleLimitsBased: \( G \times GPS \rightarrow \text{Real} \), where dist is used as a shorthand of distance

\[
dist\text{GranuleLimitsBased}(g, loc) = \begin{cases} 
0 & \text{if } \text{inGranule}(g, loc) \\
\text{distLimitsGranule}(g, loc) & \text{otherwise}
\end{cases}
\]

Besides, a showGranule operation is defined for granules, which represents the granule for a user (by using graphics, sounds, text, etc.). A similar operation, called showGranuleObjects, takes a set of objects (e.g., the objects within the granules) as an input and may represent the granule differently depending on them. For example, this could be used to represent areas with colors of different intensity depending on the traffic density. Several representation mechanisms are possible, as indicated in Section 2.1.

**Definition 3. Datatype Location Granule Map (M).**

A value of type Map (in the following, type M) is defined by a tuple \(<\text{id}, \text{granules}>\) where id is the granule map identifier and granules is the set of location granules that compose the granule map. Thus, when a granule is said to be of type \( m \), it is considered as one of the granules in the granule map identified by \( m \). However, it should be noted that, in the definition of the datatype Location Granule shown previously, there is no type attribute characterizing a location granule. This is because a granule can belong to several granule maps at the same time. Thus, granule definitions can be re-used to compose different granule maps. For example, as shown in Figure 4, if there is a granule map \( M_1 \) composed of granules corresponding to the different regions in Spain and another granule map \( M_2 \) with granules defining provinces of Spain, a new granule map where some regions in \( M_1 \) are replaced by the corresponding province granules in \( M_2 \) (if a finer location granularity within those regions is desired) can be built easily. The following operations are defined for the location granule maps:

• getGranules: \( M \times GPS \rightarrow \{ g \in G \} \subset P(G) \)

\[
\text{getGranules}(m, loc) = \{ g \mid (g \in m.\text{granules}) \land (\text{inGranule}(g, loc)) \}
\]

• getNearestGranule: \( M \times GPS \times D \rightarrow G \)

with \( D = \{ \text{distCentroidGranule, distLimitsGranule, distGranuleLimitsBased} \} \)

\[
\text{getNearestGranule}(m, loc, d) = g \mid (g \in m.\text{granules}) \land \\
\land (\exists g' \in m.\text{granules} \mid d(g', loc) < d(g, loc))
\]
Figure 4: Combining granule maps: region granules in Spain and province granules in the region of Andalucía

- \texttt{getGranulesObject: M \times O \rightarrow \{g, g \in G\} \subset P(G)}
  \[ \texttt{getGranulesObject}(m, o) = \texttt{getGranules}(m, o.\texttt{loc}) \]

- \texttt{getNearestGranuleObject: M \times O \times D \rightarrow G}
  \[ \texttt{getNearestGranuleObject}(m, o, d) = \texttt{getNearestGranule}(m, o.\texttt{loc}, d) \]

For the purpose of simplicity, an operation \texttt{map(map-id)} that returns the location granule map whose attribute \texttt{id} equals \texttt{map-id} and an operation \texttt{gr-map(map-id, gr-id)} which returns the granule whose \texttt{id} equals \texttt{gr-id} in the map identified by \texttt{map-id} are considered.

Notice that the previous operators allow mapping objects or GPS locations to the granules that contain them. In a given granule map, a GPS location could be within the areas of several granules (e.g., location granules could be defined as the coverage areas of a cellular network, where there is usually some overlapping among the cells). In such a case, \texttt{getGranules} would return all those granules and \texttt{getNearestGranule} would return only the nearest granule according to a specified distance metric \texttt{D (dist-CentroidGranule, distLimitsGranule, or distGranuleLimitsBased)}, as indicated above. However, if several granules are at the same minimum distance, one of them will be selected randomly.

Granule maps can be defined by specifying the areas of each granule (e.g., a granule map can be defined based on an SVG file\(^5\)). Different granule maps can be defined over the same geographic area, and the most appropriate one according to the context of the user can be chosen. For example, a manager of a nation-wide bus company could track the nearest city to each bus; if something strange is detected (e.g., a delay according to the expected schedule), then he/she can decide to use a granule map with a finer resolution, or even use GPS resolution.

In the current proposal, granule maps that do not evolve along time have been considered, although other proposals that focus on modeling issues have studied the case of spatio-temporal granularities (Camossi et al., 2008; Bertino et al., 2010). It is also interesting to highlight that the operations defined for granule maps in this paper allow to consider the relations between GPS locations or objects and granule maps, but not the relations between granule maps. However, the work presented in Belussi\(^6\)

\(^5\)http://www.w3.org/Graphics/SVG/, accessed March 5, 2011.
et al. (2008, 2009) could be adopted to establish relationships between different granule maps (such as containment, intersection, overlapping, subgranularities, etc.), that could later be exploited to facilitate their management and/or to perform some inferences (involving different granule maps) during the query processing.

3.2. Basic architecture

This section defines the basics of the architecture proposed for the processing of location-dependent queries with location granules. The architecture is shown in Figure 5, where the following elements can be distinguished:

- **Hardware components:**
  - One or several *mobile devices*, managed by mobile users who launch location-dependent queries. The mobile devices only perform the operations needed to interact with their users and represent the query results.
  - A *Server*, which processes the user queries and continuously refreshes the answer to those queries by communicating updates to the mobile devices, thereby alleviating the processing overload of the user devices.

For the purpose of clarity, a centralized architecture is described in this section. However, the approach presented in this paper has been developed with a distributed context in mind, with several *Servers* (e.g., database engines, not necessarily with spatial capabilities), where each *Server* manages the location data of the objects moving within a different geographic area. Indeed, the current proposal to process location granules has been tested by integrating it with a distributed location-dependent query processing system. It should be noted that an important issue in the distributed version of the architecture is the selection...
of the Servers which are relevant for a query. The distributed solution will be described in more detail in Section 5.

- **Software components:**

  - *Location Server*, a module of the Server that handles location data about moving objects and is able to answer standard SQL-like queries about them. No assumption is made about the way that this location information is managed (e.g., stored in databases, estimated by using predefined trajectories, or pulled on demand from the moving objects themselves). Thus, any *Data Management System (DMS)* could be used, such as a *Database Management System (DBMS)*.

  - *Query Processor*, a module of the Server able to process location-dependent queries with location granules by interacting with the Location Server module.

- **Data managed:**

  - *Query Table*, used by the Query Processor to store the active continuous queries. Changes in the answers to these queries must be communicated to the corresponding mobile devices with a certain refreshment frequency. However, it should be noted that a continuous query is not just a sequence of instantaneous queries executed periodically (Ilarri et al., 2006a). Thus, the Query Processor will not recompute a query from scratch at every refreshment.

  - *User Granule Maps*, defined explicitly by the user (usually a specialized or intermediate user, as a normal end-user is not expected to define granule maps).

  - *Server Granule Maps*, predefined granule maps that the user can reference in his/her queries.

  - *Granule Maps Cache*, which stores user granule maps that are used by the queries stored in the Query Table.

In this way, a user can interact with the system in two different ways: the user can launch a query that only references predefined granule maps or a query that uses granule maps which are not predefined. The first case enables considering situations where defining granule maps at the Server level is convenient. For example, a Server that stores locations about inter-city buses can have a predefined map where the granules are cities, as many users will find this map useful to build their queries. The second case leads to a more flexible querying because the users can define their own granule maps. In this case, the user-defined granule maps needed to process the query are communicated to the Query Processor along with the query.

One way of defining a granule map is to use a raster to vector converter to identify the boundaries on a geographic map. However, the user could also choose any other alternative. The user may also need to define the appropriate *showGranule* and *showGranuleObjects* operations (see Section 3.1) for visualization purposes if the query
retrieves location granules in the SELECT clause (see Section 2.1). These operations are actually performed only on the user device when the results ought to be represented (once for each location granule retrieved).

Before concluding this section it should be clarified that, for the purpose of simplicity, it is considered here that the user defines the queries to be processed. However, it should be noted that a final end-user is not expected to directly write the queries in an SQL-like query language; instead, he/she would probably use other high-level facilities. Similarly, as commented before, it is not expected that an end user would define the granule maps; these would probably be defined by more specialized/intermediate users or administrators.

4. Query processing approach

In this section, the basics of the proposed query processing approach is presented by illustrating with an appropriate example scenario. The focus is on inside constraints because other types of location-dependent constraints (e.g., nearest) can be expressed by using inside constraints (for more details, see Ilarri et al. (2006a)). The proposed algorithm is presented and described in detail in Appendix A and the complexity of the algorithm is analyzed in an electronic appendix.

The example scenario presented in Figure 6 is used. This figure shows Spain with two different granule maps overlaid: provinces of Spain (granule map with identifier “province”) and regions of Spain (granule map with identifier “region”). Thus, the gray lines mark the boundaries of provinces and the black lines indicate the boundaries of regions. Regions in Spain are composed of several provinces. For example, the figure shows how the north-east region of Spain (Cataluña) is composed of four provinces.

If location granules are associated to a location-dependent constraint, three cases can be distinguished. Thus, a location granule map can be specified for: 1) the reference object, 2) the target class, and 3) both the reference object and the target class. Sample queries for each of these cases are shown in Figure 7. In this section, the query processing approach is illustrated by considering the third case (last query in Figure 7),
as it combines the features of the previous two. Besides, an interactive demonstration showing the other cases is available as a Java applet.

<table>
<thead>
<tr>
<th>Case 1: Retrieve the identifiers of the cars within 130 miles of the province where car38 is</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT Car.id</td>
</tr>
<tr>
<td>FROM Car</td>
</tr>
<tr>
<td>WHERE inside(130 miles, gr(&quot;province&quot;, &quot;car38&quot;), Car)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case 2: Retrieve the identifiers of the cars located in regions whose boundaries are (totally or partially) within 130 miles of car38</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT Car.id</td>
</tr>
<tr>
<td>FROM Car</td>
</tr>
<tr>
<td>WHERE inside(130 miles, &quot;car38&quot;, gr(&quot;region&quot;, Car))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case 3: Retrieve the identifiers of cars located in regions whose boundaries are (totally or partially) within 130 miles of the province where car38 is</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT Car.id</td>
</tr>
<tr>
<td>FROM Car</td>
</tr>
<tr>
<td>WHERE inside(130 miles, gr(&quot;province&quot;, &quot;car38&quot;), gr(&quot;region&quot;, Car))</td>
</tr>
</tbody>
</table>

Figure 7: Sample queries with location granules

The most complicated case arises when location granules are considered for both the reference object and the target objects. In such a case, the inside constraint has the following syntax:

\[
\text{inside}(r, \text{gr}(\text{map1-id, obj-id}), \text{gr}(\text{map2-id, target}))
\]

which retrieves the target objects located in the location granules (defined by the granule map identified by map2-id) whose boundaries are within the relevant radius \( r \) from the granule of the reference object identified by obj-id (determined according to the granule map identified by map1-id). When the location of the reference object as well as the locations of the target objects have to be interpreted according to some granule map (not necessarily the same, as shown in Figure 7), the corresponding inside constraint is interpreted as follows:

\[
\text{inside}(r, \text{gr}(	ext{map1-id, obj-id}), \text{gr}(	ext{map2-id, target})) = \{ \text{oi} | (\text{oi} \in \text{target}) \land (\exists p1 \in \text{GPS} \land \exists p2 \in \text{GPS} \land \text{distance}(p1, p2) \leq r) \land \text{inGranule} (\text{gr}(	ext{map2-id, target}), p2) \land \text{inGranule} (\text{gr}(	ext{map1-id, obj-id}), p1) \}
\]

where \( \text{obj=object(obj-id), map1=map(map1-id), map2=map(map2-id)} \), and distance represents the Euclidean distance between two geographic locations.

In this case, the following operations are performed to obtain an answer (see Figure 8 for an example). First, the granule of the reference object (according to the location granule map identified by map1-id, which in the example corresponds to provinces...
of Spain) is obtained\(^7\) and the area/s corresponding to such a granule is/are enlarged by the relevant radius to obtain the relevant area/s (see Figure 8.a). Then, the set of granules in the granule map specified for the target class (i.e., the granule map identified by map2-id, which in the example corresponds to regions of Spain) that are intersected by such an area are determined (see Figure 8.b). Finally, the target objects within those granules (enlarged relevant area) are retrieved (see Figure 8.c). The operation corresponding to the first step implies computing the Minkowski sum of the area/s composing the location granule of the reference object and a disk with radius the relevant radius (the Minkowski sum of two sets in the Euclidean space is obtained by adding every element of one set to every element of the other set). This operation is called buffering in the context of Geographic Information Systems (van Kreveld, 2006):

\[
\text{buffer}(r, \text{granule}) = \text{granule}' | (|\text{granule}.A | = |\text{granule}'.A |) \land \\
(\forall A_i' \in \text{granule}'.A: \exists A_i \in \text{granule}.A | A_i' = \text{buffer}(r, A_i))
\]

\[
\text{buffer}(r, \text{area}) = \text{area}' | (\forall p \in \text{GPS}: \text{contains}(\text{area}, p) \implies \text{containsA}(\text{area}', \text{circle}(p, r)))
\]

\[
\text{containsA}(\text{area1}, \text{area2}) \iff (\forall p \in \text{GPS} | \text{contains}(\text{area2}, p) \implies \text{contains}(\text{area1}, p))
\]

Figure 8: Granules for the reference object and the target objects: (a) obtaining the relevant area, (b) obtaining the granules that are intersected, and (c) retrieving the target objects within those granules

5. Integration with a distributed location-dependent query processing system

In this section, the proposed architecture and query processing approach to handle location granules is studied within the context of LOQOMOTION (Ilarri et al., 2006a), a pre-existing system developed for the efficient processing of continuous location-dependent queries over a distributed infrastructure of computers. In LOQOMOTION, these computers are called proxies, such that each proxy handles the location data of the objects moving within its proxy area and is based on a certain Data Management System (DMS), such as a relational database. Thus, the concept of proxy in LOQOMOTION is similar to the concept of a Server presented in Section 3.2, but in a distributed

\(^7\)For simplicity, and without loss of generality, two assumptions have been made in the text explanation. Firstly, although getGranulesObject can obtain several granules (see Section 3.1), it has been considered that a single granule is returned. Secondly, it is also assumed that every object is within some granule.
environment. In the rest of this section, the architecture of LOQOMOTION is introduced, emphasizing the aspects that have been added to enable the processing of queries with location granules. LOQOMOTION is based on a hierarchical architecture of mobile agents (Trillo et al., 2007) that deploy themselves on the fixed infrastructure to process the queries and keep the answers up-to-date. First, the architecture of LOQOMOTION is summarized. Then, the basic aspects of the query processing in relation to the use of location granules are described. Finally, the flexibility of the architecture to process other types of location-dependent constraints is emphasized. Some modifications of the main approach presented in this paper to process queries with location granules (formalized in the algorithm included in Appendix A, see Figure A.1) are needed to adapt to LOQOMOTION and the distributed infrastructure under consideration (e.g., different tasks must be performed distributively on several proxies).

5.1. Basics of the architecture of LOQOMOTION

In this section, the basics of the agent-based architecture of LOQOMOTION without considering location granules are briefly described. No attempt is made to justify the use of mobile agents or the relation between LOQOMOTION and other systems for location-dependent query processing (for details about this, see Ilarri et al. (2006a) and/or the survey in Ilarri et al. (2010)). The purpose of this section is only to explain some basic aspects that are needed in order to understand how location granules are later integrated in the system. To clarify the explanations, the scenario shown in Figure 9, and a query that retrieves the interesting objects within the (inner) moving query circles (relevant areas) centered on the reference objects car38 and policeCar5, will be considered. In LOQOMOTION there is a static agent called QueryMonitor, executing on the mobile device of the user, and three different types of agents (two of them, mobile agents (Trillo et al., 2007)) are in charge of processing the query on the fixed network:

- A mobile agent MonitorTracker on the fixed network (initially, on Proxy6 in Figure 9) is in charge of communicating, to the user device, the updated data about moving objects relevant to the query (target objects). This agent always executes on the proxy corresponding to the proxy area where the user is, following the user, to optimize communications with the mobile user device.

- For each reference object, a mobile agent Tracker keeps itself on the proxy that handles the location of that reference object (e.g., in Figure 9 the Tracker for car38 is initially executing on Proxy4, as car38 is within the coverage of that proxy), to track it, and computes the area that contains the objects that must be communicated to the MonitorTracker (called extended area).

- For each proxy whose area intersects the extended area (the relevant proxies), a static Updater agent is created by such a Tracker (e.g., in Figure 9, the Tracker on Proxy4 sends Updaters to Proxy3, Proxy4, and Proxy5). An Updater is in

---

8The areas shown in Figure 9 are circles, but this will not be necessarily the case when location granules are used, as will be explained in Section 5.2.
charge of retrieving the interesting objects within its area by executing *standard queries* (i.e., queries without location-dependent constraints such as *inside*) on its proxy.

![Diagram of query processing in LOQOMOTION: sample scenario](image)

The different mobile agents maintain themselves on the relevant proxies, to keep track of the interesting objects. In this way, they support an efficient continuous query processing, as results can be ready when a refreshment is needed. Summing up, the use of mobile agents in LOQOMOTION facilitates: 1) tracking the positions of relevant objects efficiently, 2) optimizing the wireless communications, and 3) supporting the distributed query processing efficiently. For details about this, see Ilarri et al. (2006a).

An applet illustrating the basic aspects of the architecture of LOQOMOTION is available in the Web. Some additions are needed in the initial architecture to enable the processing of queries with location granules. Thus, every agent should be able to consider the locations of the moving objects in terms of the types of location granules specified by the user in the query constraints. If user granule maps are used (see Section 3.2), such maps must be made available to the different agents involved in the query processing (by propagating the granule maps through the network of agents).

Location granules can be managed in LOQOMOTION using the approach described in this paper. By integrating it with LOQOMOTION, continuous queries with

---

9It should be noted that, as indicated in Section 3.2, this is not equivalent to a sequence of instantaneous queries that are executed periodically.

10See [http://sid.cps.unizar.es/SILARRI/LOQOMOTION-APPLET/AgentDeploy.html](http://sid.cps.unizar.es/SILARRI/LOQOMOTION-APPLET/AgentDeploy.html).
location granules can be managed in a distributed environment, which is highly beneficial for the scalability of the system.

The rest of this section explains some aspects of the query processing of LOQOMOTION where the use of location granules has to be considered: computing the areas of interest, obtaining location data of the target objects and composing an answer, and keeping the results up-to-date.

5.2. Computing the areas of interest

Every Tracker is in charge of processing the constraints where its reference object appears. From each one, it obtains a relevant radius (which indicates the radius relevant to the constraint) and a radius extension (by considering certain parameters such as the maximum speed of the relevant objects). With the above mentioned data and the current location of the tracked reference object (which the Tracker obtains by querying its proxy), the Tracker determines the geographic area which is relevant to its constraint (the relevant area). By enlarging the relevant area with the radius extension, the Tracker obtains an extended area. Keeping track of all the objects within the extended area, instead of only those within the relevant area, increases the fault-tolerance of the system when a location prediction mechanism is used (Ilarri et al., 2006a).

If location granule maps are not used in the constraint, the relevant and extended areas are circles centered on the reference object with radius the relevant radius and the sum of the relevant radius and the radius extension, respectively. On the contrary, if a location granule must be associated to the reference object of the constraint, the Tracker considers the current location granule of the reference object and extends it in all directions by the relevant radius to obtain the relevant area (see Figure 8.a). Then, the extended area is computed by extending the relevant area according to the radius extension.

The computation of these areas is therefore consistent with the explanation in Appendix A (lines 1-14 of Algorithm 1). The main difference is that extended areas, and not only relevant areas, are considered in LOQOMOTION for the purpose of reliability. Besides, in LOQOMOTION this computation is performed by a Tracker agent on the proxy that covers the area where the reference object is located, instead of directly by a Query Processor module on a Server (see Section 3.2). This is because the Tracker adopts the role of Query Processor.

5.3. Obtaining location data of the target objects and composing an answer

Each Tracker must determine the set of proxies relevant to its query constraints (i.e., the proxies in charge of managing the location data about the target objects). From the extended area obtained in the previous step, the set of relevant proxies is computed by considering whether a location granule map has been specified in the constraint for the corresponding target class or not. To accomplish this, the extended area is modified, if necessary, to obtain an enlarged extended area. Thus, if there is a granule map specified for the target class, the corresponding granule map is used to obtain the

---

11In an inside constraint such as inside(0.42 miles, "policeCar5", policeCar), this relevant radius (0.42 miles) appears explicitly in the constraint.
granules that intersect the extended area, and the union of these granules determines
the enlarged extended area. If there is no granule map specified for the target class,
then the enlarged extended area is just the extended area. This is consistent with the
explanation in Appendix A (lines 15-18 of Algorithm 1). The relevant proxies are
obtained as the proxies whose proxy area intersects the enlarged extended area. The
Tracker creates an Updater on each of these proxies.

The corresponding Updaters execute standard queries that retrieve the interesting
objects within the enlarged extended area computed by their Tracker (lines 19-29 of
Algorithm 1 in Appendix A). If no location granule map is specified either for the
reference object or the target class, this area will be a circle and the query can be easily
executed, for example, on a proxy with a relational database. If not, then a query
that asks about the objects within the MBR (Minimum Bounding Rectangle) of the
extended area computed is considered (filter step). Then, a refinement step is performed
to remove any false positives. Therefore, a Tracker must communicate that query and
the enlarged extended area to its Updaters (so that each Updater is able to remove
the false positives that it obtains).

The different types of agents cooperatively obtain the relevant data that are finally
sent to the mobile device of the user, where the granule maps specified in the SELECT
clause are used to represent the retrieved locations according to the user’s needs, as
explained in Section 2.1.

5.4. Keeping the results up-to-date

As objects are continuously moving, the initial results obtained become obsolete in
a short time. LOQOMOTION considers the processing of continuous queries, which
means that the query must be processed continually until it is canceled by the user.
For this purpose, the system must perform two related tasks: 1) to keep the network of
agents ready on the relevant proxies, according to the current locations of the reference
objects; and 2) to update the answer to the query automatically. The above mentioned
two tasks are explained in the rest of this section.

5.4.1. Task 1: Maintenance of the Network of Agents

While a continuous query is active (i.e., it has not been canceled by the user),
the deployed network of agents must adapt to changes in the locations of the moving
objects relevant to the query. For example, the MonitorTracker must move in order
to keep itself close to the user. Similarly, Trackers must move according to the locations
of their reference objects. Moreover, Trackers must rearrange their Updaters as needed to
make them watch their extended areas, as these areas move with their reference objects.
The following part describes in detail how the Trackers contribute to maintaining the
network of agents ready to retrieve the required data.

As the reference object moves, the shape and location of the enlarged extended
area associated to that reference object can change. For this reason, every Tracker
must obtain the location of its reference object with a certain tracking frequency and

\[\text{12}\text{The use of MBRs and the refinement step can be avoided if the proxy supports queries about objects}
\text{within areas with arbitrary shape. For more details, see Appendix A.3.}\]
recompute the enlarged extended area as needed. A change in this area can have two implications:

- The standard queries executed by the Updaters become out-of-date, as those queries retrieve the objects within the old enlarged extended area (computed by the corresponding Tracker based on a previous location of its reference object). For this reason, the Tracker must communicate the new standard queries / enlarged extended areas to their Updaters.

- The set of relevant proxies may change. The Tracker would obtain the new proxies whose proxy areas intersect the new enlarged extended area, to determine the proxies where an Updater is needed. If the set of proxies relevant to the Tracker changes, the Tracker would need to rearrange its network of Updaters, that is, it will request its unnecessary Updaters to finish and create new Updaters if needed. Keeping the network of Updaters on the relevant proxies is a key aspect to ensure that the interesting geographical area can be watched. It can be noted that the use of extended areas, besides increasing the fault-tolerance of the system (see Section 5.2), allows to anticipate the need of new Updaters to watch the areas of interest.

A simple example illustrating how the network of Updaters is maintained can be seen in Figure 10. On the top part of the figure a scenario with three proxies is shown and four steps are considered during the movement of a reference object \textit{refObject}. On the bottom part of the figure the agents executing on each proxy at each step can be seen. At Step 1 the MonitorTracker is executing on \textit{Proxy3} (close to the user) and the Tracker is executing on \textit{Proxy1} (as \textit{refObject} is within its proxy area), and there is an Updater on \textit{Proxy1} and another one on \textit{Proxy3} (as the enlarged extended area intersects their proxy areas). At Step 2 the Tracker is executing on \textit{Proxy2} (as the reference object is then within its proxy area) and a new Updater has been created by the Tracker on \textit{Proxy2} (as the network of Updaters must be able to watch the whole enlarged extended area). At Step 3 the Updater on \textit{Proxy1} has been removed, as it is not required anymore (the enlarged extended area does not intersect the proxy area of \textit{Proxy1} anymore). Finally, at Step 4 the Updater on \textit{Proxy3} has also been removed (only an Updater on \textit{Proxy2} is maintained).

The way in which the use of location granules affects the tasks related to the maintenance of the network of agents can be observed here. If a location granule map has been specified for the reference object, its location (and therefore the extended area), from the point of view of the query processing, will not change unless the object moves to another granule. Consequently, the tasks related to the maintenance of the network of Updaters are performed by the Tracker less frequently, thereby reducing the overhead of this part of the query processing. To illustrate this, a simple example is shown in Figure 11. In this case, \textit{refObj} does not exit its granule along the movement shown in the figure. Therefore, the network of Updaters does not change. The only action performed is the migration by the Tracker from \textit{Proxy1} to \textit{Proxy2} at Step 2. This is because the reference object is within the proxy area of \textit{Proxy2}; therefore, only \textit{Proxy2} can be queried about the location of that reference object. Similarly, the cost of maintaining the network of Updaters also decreases when a location granule map has been
Figure 10: Maintenance of the network of agents: no granule map is specified for the reference object

specified for the target class. This is because only a change in the intersection between the proxy areas and the relevant granules in the granule map will lead to a change in the set of relevant proxies.

Moreover, the tracking frequency of a Tracker is adapted automatically when a location granule map has been specified for its reference object. Thus, by assuming a maximum speed for the reference object (an upper-bound can be considered for each type of object) the Tracker can compute the minimum amount of time needed by the reference object to move to another granule. During that interval it is guaranteed that the location granule of the reference object will not change. Thus, the enlarged extended area will not change either. Consequently, during that interval there is no need to rearrange the network of Updaters. On the contrary, when the reference object is moving next to the border of a granule, the Tracker automatically increases its tracking frequency to detect the change of granule as soon as possible. This strategy greatly minimizes the possibility of incurring a significant error in the answer obtained to the query due to the consideration of an out-of-date location granule for the reference object.

The tracking frequency considered by the Tracker is actually obtained as the inverse of $\max(\min(T_{exitGranule}, T_{exitProxyArea}), T_{refreshment}) - \text{securityMargin}$, where $T_{exitGranule}$
is the minimum amount of time needed for the reference object to exit the granule, $T_{exitProxyArea}$ is the minimum amount of time required for the reference object to exit its current proxy area, $T_{refreshment}$ is the current query refreshment period (i.e., the inverse of the query refreshment frequency), and $securityMargin$ is a small amount of time that can be added to account for slight inaccuracies in the previous estimations. $T_{exitProxyArea}$ is considered because, even if the network of Updaters does not require to be rearranged, the Tracker must follow its reference object from proxy to proxy. As the previous formula shows, if the $securityMargin$ is left aside, the maximum tracking frequency considered is the same as the query refreshment frequency. To keep the network of Updaters ready for the next refreshment, the moment when the tracking is performed is determined by taking into account the time instant when the Tracker must communicate the new results to its MonitorTracker (i.e., the deadline of the Tracker, see Section 5.4.2) and the delays involved.

5.4.2. Task 2: Periodic Refreshment of the Answer

Data obtained by the QueryMonitor about the target objects could become obsolete whenever any target or reference object moves to another location. As the objects move
continuously, and LOQOMOTION considers the processing of continuous queries, the Updaters must execute their queries with a certain refreshment frequency (e.g., once every five seconds) specified by the user or automatically determined by the system (Ilarri et al., 2008). In general, the continuous query processing implies that all the agents in the architecture of LOQOMOTION must obtain new data with the required refreshment frequency and synchronize among themselves to correlate the data obtained from the distributed set of proxies involved in the query. For more details about the synchronization issues, which are independent of the use of location granules, see Ilarri et al. (2008). In particular, in the above mentioned work a synchronization mechanism based on the assignment of deadlines to agents has been proposed, by which every agent tries to return, with the required frequency, data as recent as possible to its creator agent.

When the user specifies that the locations of the objects that have to be projected are granules defined in a certain granule map, some communications can be saved by using an incremental update of the answer on the mobile device (see Section 6.3 for an experimental evaluation showing this). The idea is that only the updates to the set of objects sent previously to the mobile device must be communicated. Thus, if some object did not move to another location granule, there is no need to communicate that data again. Obviously, this incremental approach is useless when the precise geographic coordinates of the objects must be retrieved (as these will change continuously). Intuitively, it is clear that more wireless communications will be required if the mobile user is interested in the precise geographic locations of the moving objects.

5.5. Summary of implications of granules for the continuous query processing

The implications of using location granules in the continuous query processing (see Table 1) are summarized below:

<table>
<thead>
<tr>
<th>Specifying a granule map for...</th>
<th>Implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>... the reference object</td>
<td>- Enlarged extended area obtained by buffering the location granule of the reference object.</td>
</tr>
<tr>
<td></td>
<td>- Less overhead to maintain the network of Updaters.</td>
</tr>
<tr>
<td></td>
<td>- Adaptive tracking frequency for the Tracker.</td>
</tr>
<tr>
<td>... the target class</td>
<td>- Enlarged extended area obtained by intersecting a circle with the granules in the granule map.</td>
</tr>
<tr>
<td></td>
<td>- Less overhead to maintain the network of Updaters.</td>
</tr>
<tr>
<td>... both the reference object (map1) and the target class (map2)</td>
<td>- Enlarged extended area obtained by buffering the location granule of the reference object (according to map1) and intersecting the resulting area and the granules in the granule map for the target class (map2).</td>
</tr>
<tr>
<td></td>
<td>- Minimum overhead to maintain the network of Updaters.</td>
</tr>
<tr>
<td></td>
<td>- Adaptive tracking frequency for the Tracker.</td>
</tr>
<tr>
<td>... a projection (SELECT clause)</td>
<td>- Less wireless communications by using an incremental update policy.</td>
</tr>
</tbody>
</table>

Table 1: Impact of using location granules
When a location granule map has been specified for the reference object, the enlarged extended area associated to the constraint turns into an arbitrary shape (instead of being a circle around the reference object), depending on the shape of the location granule where the reference object is located (as indicated in Section 2.2 the reference can also be directly a location granule). This area is obtained by enlarging that location granule according to the relevant radius (buffering operation). Besides, as the computation of the relevant area (and thus the enlarged extended area) is based on that location granule instead of a precise GPS location, fewer changes in the network of deployed Updaters are necessary. Finally, the tracking frequency of the Tracker is adaptively modified: in general, it is reduced when the reference object is far from the boundary of the granule and increased when it is near the border.

When a location granule map has been specified for the target class, the enlarged extended area associated to the constraint has now a shape determined by the union of some location granules in the granule map specified for the target class. Thus, it is computed by intersecting a circle (with radius the relevant radius and centered on the current GPS location of the reference object) and the granules in that location granule map. Moreover, the overhead of maintaining the network of Updaters will decrease, as the set of relevant proxies will change less frequently (only when the intersection between the area defined by the intersected granules in the location granule map and the proxy areas changes).

When a location granule map has been specified for both the reference object and the target class, a mixture of the two previous situations is obtained. Thus, the enlarged extended area is computed by intersecting the area obtained by buffering the location granule of the reference object and the granules in the granule map specified for the target class. From the point of view of the overhead of the Updaters’ network maintenance, this case is even more stable than the previous ones, as the set of relevant proxies changes even less frequently. The tracking frequency of the Tracker is also adaptively modified.

When location granules (instead of precise geographic locations) are projected in the queries, the amount of wireless communications with the mobile device is reduced by using an incremental update of the answer (i.e., by communicating only the changes in the answer). Thus, the location granule where an object is located is expected to change much less frequently than its precise GPS location.

As described, a clear implication of specifying location granule maps in the constraints of queries is the change in the way the relevant area is computed. However, there are also other implications. Thus, the use of location granules also reduces the overhead of some query processing tasks (such as the effort needed to maintain the network of Updaters) and the amount of wireless communications when the location granules are projected.

5.6. Processing other types of constraints with the architecture of LOQOMOTION

As indicated in Section 4, so far the focus has been on the processing of inside constraints. However, the architecture of LOQOMOTION also supports the processing
of other location-dependent constraints with location granules. In this section, the flex-
ability of the proposal to process join queries, aggregate queries, and nearest-neighbor
queries is briefly highlighted. It is important to emphasize that there is no intention to
provide elaborated solutions for the processing of these kinds of queries, but only to
show how they can be easily processed with the proposed architecture.

5.6.1. Location-dependent joins with location granules

So far, in this paper queries consisting of a single location-dependent constraint
have been presented. However, more than one location-dependent constraint can ap-
pear in a query. Moreover, there may be a condition linking the tuples satisfying
two different location-dependent constraints. That is, there could be a join condi-
tion (Mishra and Eich, 1992). The proposed architecture can also manage queries with
joins.

As an example of a join query, assume that a taxi company wants to retrieve
couples <person, taxi>, such that person is a person looking for a taxi and located
within 1.2 miles from the city center, and taxi is an available taxi within 1.5 miles from
the city center and at a maximum distance of 0.3 miles from the potential customer (see
Figure 12).

```
SELECT Person.id, Taxi.id, distance(Person, Taxi)
FROM Person, Taxi
WHERE inside(1.5 miles, gr-map("neighborhoods", "CityCenter"), Taxi) and
    inside(1.2 miles, gr-map("neighborhoods", "CityCenter"), Person) and
distance(Person, Taxi) <= 0.3 miles
```

Figure 12: Sample location-dependent join query with location granules

where an operator distance that computes the Euclidean distance between two objects
of the specified classes is considered.

A join query like the one presented in Figure 12 is processed as follows. First,
the two inside constraints are processed at the Tracker level as usual. The instances of
Taxi and Person satisfying these constraints are returned to the MonitorTracker: more
specifically, the id attributes (needed for the projection in the SELECT) and the loc
attributes (needed to compute the distance) are communicated to the MonitorTracker.
Then, the MonitorTracker computes the join by applying the distance constraint. Fi-
ally, the results are sent from the MonitorTracker (executing on a fixed proxy) to the
QueryMonitor (executing on the mobile device of the user).

5.6.2. Aggregate functions in location-dependent queries with location granules

In many applications, computing aggregated values is of great importance. As de-
defined in Klug (1982), “An aggregate function takes a set of tuples (a relation) as an
argument and produces a single simple value (usually a number) as a result.” Five pop-
ular and widely-used aggregation functions, defined in the standard SQL-92, are (Gray
et al., 1997): COUNT, SUM, MIN, MAX, and AVG.

The architecture described in the current paper supports projecting aggregates in
location-dependent queries with location granules. In particular, so far the aggregate
functions defined in SQL-92 have been considered. For example, the query in Figure 13 obtains the number of cars within 130 miles around the province where car38 is located. Aggregate queries like the one presented in the example can be processed by simply applying the aggregate function once the result for the query without the aggregates has been obtained. Therefore, in the proposed architecture the aggregate computation is performed by the MonitorTracker before sending the final results to the user. In this way, the amount of information transferred through the wireless medium (from the MonitorTracker to the QueryMonitor) is minimized (only the aggregated value computed, and not the whole tuple set, must be communicated). Even though in some cases (like in the one presented in Figure 13) it would be possible to compute the aggregate at the Tracker level, this is not generally true (e.g., the tuples used to compute the aggregate may be the result of a join of tuples obtained by two Trackers).

```sql
SELECT COUNT(Car.id) FROM Car WHERE inside(130 miles, gr("province", "car38"), Car)
```

Figure 13: Sample location-dependent aggregate query with location granules

The cost of the above mentioned approach is similar to the cost of processing the corresponding location-dependent (inside) constraint (with the extra overhead of computing the aggregate). More sophisticated approaches could be considered to compute aggregates. Indeed, there has been an intensive research effort in devising efficient aggregate computation methods for different scenarios (see Ilarri et al. (2010)). However, the aim here is just to show how the proposed architecture can flexibly process them.

5.6.3. Nearest neighbor queries with location granules

As a final case, nearest constraints are also supported through the processing of inside constraints. As an example of location-dependent nearest neighbor query with location granules, the query in Figure 14 would retrieve the five cars that are the nearest ones to the current province of car38.

```sql
SELECT Car.id FROM Car WHERE nearest(5, gr("province", "car38"), Car)
```

Figure 14: Sample location-dependent nearest neighbor query with location granules

To obtain the objects that satisfy a nearest constraint with a granule mapping specified for the reference object, like in the query presented in Figure 14, the objects of the target class located within the granule of the reference object are retrieved. These objects are closer to the granule than any other object outside the granule, as they are at distance zero. Therefore, if the number $M$ of objects retrieved is not smaller than the number $N$ of nearest objects to retrieve, a subset of $N$ of those objects can be directly returned as the answer to the query. Otherwise, an iterative technique is applied to retrieve $N' = N - M$ additional objects. The technique is inspired by the proposal in
the study of Jagadish et al. (2005), which retrieves the objects within a query sphere that increases iteratively until all the nearest objects required are obtained. Thus, to retrieve the additional objects required, a buffering operation is applied to the granule by extending it by an amount \( r' \) and the objects inside are collected. If there are still not enough objects to answer the nearest query, the operation is repeated, and so on. Once the enough number of objects has been collected, the objects are sorted according to their distances to the granule of the reference object and the first \( N \) objects are returned. A similar procedure is applied when there is a location granule map specified for the target class or for both the reference object and the target class. The iterative process mentioned can be performed by the corresponding Tracker agent, which relies on its Updaters to retrieve the objects within the query sphere considered in each iteration. For more details about the processing of nearest constraints, see Ilarri et al. (2009).

6. Experimental evaluation

Firstly, this section describes some aspects of the prototype that has been developed to test and evaluate the ideas presented in the current work. Secondly, the experimental settings are described. Thirdly, this section evaluates how the use of location granules can minimize the amount of wireless communications performed. Fourthly, the accuracy of the query processing approach with location granules is analyzed. Lastly, the overhead introduced in the query processing due to the use of location granules is studied. Additional experiments have been included in an electronic appendix.

6.1. Prototype implemented

In the prototype developed, the following technologies have been used: Java 1.6 as the programming language, the JTS Topology Suite\(^{13}\) to perform the required geometry operations (e.g., dilation of polygons), and MySQL (DuBois, 2009) databases as the Data Management Systems (DMSs) to store the data about objects. It should be noted that the datatypes defined in Section 3.1, and most of their operators (except those related to the representation of the datatypes, such as showGranule and showGranuleObjects), could have been implemented using a DBMS with spatial capabilities, such as Oracle (http://www.oracle.com) or PostgreSQL (http://www.postgresql.org/) with PostGIS (http://postgis.refractions.net/). As the existing literature indicates (Rigaux et al., 2002; Yeung and Hall, 2007), a spatial database could provide additional advantages, such as a high scalability and support for the management of a huge amount of spatial granularities uniformly, using a single repository to store all the data (both the granularity-related data and the data about the moving objects). However, in the current prototype it was preferred to impose minimal requirements on the DMS side by relying on the common capabilities of the existing DBMSs (using standard SQL), which would allow to switch to different DBMSs without changes in the code used to access the database or perform tests in heterogeneous environments with different types of DBMSs. On the contrary, for example, different spatial DBMSs provide different level of support for OpenGIS (http://www.opengeospatial.org/).

Nevertheless, it is important to emphasize that not using the spatial DBMSs is a decision related to the prototype developed, as the proposed query processing architecture also works with spatial databases (in that case, some geometric operations, such as the **buffering operations** mentioned in Section 4, would be performed by the spatial DBMSs).

The developed prototype processes continuous location-dependent queries with location granules following the approach described in the current paper. The approach has been integrated in LOQOMOTION, as described in Section 5. LOQOMOTION uses the mobile agent platform SPRINGS (Ilarri et al., 2006b) as middleware for efficient query processing in distributed environments. In the prototype, two default types of representations for the granules selected in queries have been implemented:

- **A basic graphical representation.** Granules can be associated to colors. With this representation, the areas of a granule containing an interesting object would be colored. The interesting objects inside the granule would be represented by painting a small circle at some point within the granule with the name of the object next to it (as shown in the example of Figure 15.a).

- **A sound-based representation.** Granules can be associated to sounds, such that when an interesting object is within the granule, a certain sound will be played. For example, a train tracking application could generate a ring sound with increasing frequency as the distance to the destination decreases, to alert the user about a train that is approaching its final stop.

Moreover, new types of representations can be easily implemented by extending, with any wanted behavior, the Java classes corresponding to the previous types. Thus, the programmer has to only redefine the appropriate representation operations (**showGranule** and **showGranuleObjects**, as indicated in Section 3.1). As an example, in Figure 15.a, a snapshot of the query processor for a query that continuously retrieves buses in provinces within 130 miles of a certain bus is shown. In Figure 15.b, two bus routes are considered and the user is interested in the nearest stop to each bus. In this figure, a graphical representation different to the two alternatives mentioned above has been used. Specifically, each bus is displayed on the nearest city within its route, which is presented schematically. It should be stressed that, although the same granule map (provinces of Spain) is used in both the cases (Figure 15.a and Figure 15.b), the results are presented to the user in a different way (a different implementation of **showGranule** is considered). An interactive demonstration of other types of representations is available as a Java applet\(^\text{14}\).

### 6.2. Experimental settings

All the tests have been performed by simulating six proxies on five computers Pentium IV 2.4 GHz (2 cores) with 2 GB RAM, and connected through a 100 Mbps Ethernet. As in Ilarri et al. (2006a), the wireless communications with the mobile device

\(^{14}\text{See the second applet and/or the second demo video available at http://sid.cps.unizar.es/LOQOMOTION/loqomotion-granules.html.}\)
of the user have been simulated by delaying such communications according to the size of the data sent and a bandwidth of 40 kbps, plus a random increase of up to one second to roughly simulate unreliability. In this experimental environment, tests were performed with both real and synthetic (i.e., automatically generated) granule maps. A real granule map (e.g., defined by a map of provinces) would allow for evaluating how the prototype would behave in an existing scenario. A large set of synthetic granule maps would allow for extracting general conclusions which are independent of a specific scenario. Using a specific granule map for the experimental evaluation (even if such a granule map is real) would make it really difficult to draw general conclusions (the results could be tied to that specific granule map). Therefore, in the following experimental evaluation the focus is on the case of synthetic scenarios.

The synthetic granule maps generated for the experimental evaluation have a size of 4.5 squared kilometers. For each generated granule map, a set of granules that cover this area needs to be defined. For this, a technique based on building Voronoi diagrams from a set of random points was used. In order to make the experiments more challenging, at least 40% of the initial points of the Voronoi tessellation are located in the 35% of the central part of the scenario. This implies the generation of “stress

---

15In a separate electronic appendix an example of test with a real scenario is shown.
zones”, where there is a high number of location granules\(^\text{16}\). Images of the whole set of the synthetic granule maps generated for the experimental evaluation are available in the Web\(^\text{17}\).

In the following tests, objects moving at 50 km/h following a random path in the Euclidean space (specifically, the direction of an object changes every five seconds with a 0.3 probability) were considered. It should be noted that a road-based mobility model could have been used. However, having objects whose movements are not constrained to a road network makes the experiments even more challenging. Each test implies the processing of a query with an \textit{inside} constraint with the specification of some granule map/s; the \textit{relevant radius} \(r\) of the \textit{inside} constraint is 550 meters. This query is executed during 7 minutes with a default refreshment period of 10 seconds. Depending on the specific experiment considered, the number of moving objects in the scenario (from 100 to 500 objects, with increments of 100 objects) or the number of location granules (from 10 to 50 granules, with increments of 20 granules) may be varied to evaluate its impact. Each test is repeated several times (at least 10) and the average values are reported\(^\text{18}\).

6.3. Evaluation of the communication savings thanks to the use of location granules in the query projection

This section shows how the use of an incremental schema to communicate the changes in the answer set to the mobile device of the user (as mentioned in Section 5.4.2) allows the system to save wireless communications when granule maps are specified in the SELECT clause.

The results with 500 objects and for different numbers of location granules are shown in Figure 16. In Figure 17, 50 granules were considered and the number of moving objects was varied. Both figures show that the use of location granules together with incremental updates help to reduce the amount of data that should otherwise be communicated to the mobile device. Figure 16 also shows that, when no granule maps are specified in the query projection or the incremental update is not used, the amount of data to communicate decreases with the number of granules. This is because, for a fixed scenario size, a higher number of granules lead to smaller granules. Therefore, the number of objects satisfying the query constraints decreases. However, when granule maps are specified in the query projection and incremental updates are used, the number of updates needed does not depend on the number of granules but on the trajectories of the objects relative to the granules in the scenario (only the changes between the granules are communicated with the incremental schema). As an additional (and expected) conclusion, it is noticed that the total amount of data to communicate grows with the number of objects (see Figure 17), as the number of objects that may be relevant for the query also grows in this case.

\(^{16}\)Therefore, in this area the moving objects will change from one location granule to another more frequently.

\(^{17}\)See http://sid.cps.unizar.es/LOQOMOTION/granulae/scenarios/.

\(^{18}\)No significant differences have been observed between the results of different repetitions.
6.4. Evaluation of the accuracy in the answer set

This section measures the accuracy of the answer set provided to the user along time by showing the ideal number of objects in the answer (the objects that should be returned if a precise snapshot of the whole scenario were available with no delay), the number of extra objects (false positives, i.e., objects that are shown as part of the answer but they should not be part of the answer according to the current real situation), and the number of missing objects (false negatives, i.e., objects that are not in the answer set but should be part of the answer). Instead of showing the temporal evolution, the average values per second are reported.

The results in a scenario with 500 objects and different numbers of granules is shown in Figure 18, and the results considering 50 granules and varying the number of moving objects in the scenario is shown in Figure 19. In these figures, the middle line in the figure represents the number of objects in the ideal (i.e., correct) answer, the difference between the upper line and the middle line indicates the number of extra objects, and the different between the middle line and the lower line indicates the number of missing objects. For example, the following results are observed for the case of 500 objects, 30 granules, and granule maps for both the reference object and the target class (see Figure 18, bottom): the number of ideal objects is 141, the number of extra objects is 22 (i.e., the line marking the answer with extra objects goes through the Y-point 163), and the number of missing objects is 18 (i.e., the line marking the an-
Figure 17: Communication savings thanks to the use of location granules: impact of the number of objects

swer with missing objects goes through the Y-point 123). As mentioned before, these are the average values of observations made every second during the whole duration of the experiment. The extra and missing objects usually correspond to objects whose positions are near a granule boundary and whose movements change the ideal answer between two updates of the answer on the mobile device; such movements introduce an error that will not be corrected until the next refreshment of the answer. The existence of processing and communication delays (especially, wireless communication delays) is also a factor that can lead to inaccuracies in the answer set. As Figure 19 shows, the errors increase with the number of objects in the scenario (the separation between the line indicating the ideal number of objects and the lines indicating the number of extra and missing objects grows). This is because with a higher number of objects it is more probable that some of them are near the boundaries of the granules.

The results are considered very satisfactory. It should be emphasized that some errors are unavoidable because the objects continue moving between two refreshments of the answer. However, there are minimum errors at the refreshment time instants. Moreover, in real scenarios the movements of the objects are not expected to be so random and the size and shapes of the location granules would probably have well-defined and significant shapes, thereby contributing to further reducing the errors.
6.5. Evaluation of the processing overhead due to the use of location granules

Previous studies have shown that the basic approach of LOQOMOTION exhibited a good performance (Ilarri et al., 2006a). However, with the extension proposed in the current paper to support location granules, the fact to be taken into account is that the management of location granules during the query processing introduces a certain overhead (extra processing time due to the use of location granules). Thus, for example, some extra geometric computations must be performed when granules are used, as explained along the paper. This section measures the accumulated overhead during the 7-minute experiment, aggregated by a type of agent in LOQOMOTION (see Section 5.1) and in total. It distinguishes whether a granule map is specified for the reference object or for both the reference object and the target class. However, to save space, the case of using a granule map only for the target class has been omitted, as similar conclusions can be obtained in that case.

Figure 20 shows the results for a scenario with 500 objects and different numbers of granules, and Figure 21 has been obtained in a scenario with 50 granules and different numbers of moving objects. In general, the overhead does not increase with the number of granules. The reason is that the average size of the granules is inversely proportional to the number of granules for a fixed scenario size. Thus, increasing the number of granules also leads to granules that are smaller, and the system would need to check...
Figure 19: Accuracy in the answer set (with 50 granules in the maps): impact of the number of objects

a smaller number of objects as potential candidates that may be within the granule, thereby reducing the overhead. Nevertheless, a higher number of granules may slightly increase the overhead when there is a granule map specified for the target class, as many granules can intersect the target area (and those granules must be examined). Regarding Figure 21, as expected, the overhead increases with the number of objects (as more objects are candidates to be part of the answer to the query).

As both Figure 20 and Figure 21 show, the overhead due to the use of location granules is very small. It should be emphasized that the total cost during the 7-minute experiment and with a refreshment period of just 10 seconds has been shown in these figures (the total overhead per refreshment is always under 1 second and usually under 0.5 seconds). Moreover, just a small part of the processing overhead involves the mobile device of the user (specifically, the overhead corresponding to the QueryMonitor agent); the rest of the overhead affects only the agents executing on proxies in the fixed network. Finally, due to the distributed query processing approach, the overhead affecting the fixed network is distributed among different computers (and so the overhead per computer is even smaller). A theoretical analysis of the cost of dealing with location granules is presented in an electronic appendix.
7. Related work

To the best of the authors’ knowledge, no other work in the field of location-dependent query processing supports enhancing the expressivity of the queries through the specification of queries with different location granularities. So, other approaches do not cover the use of location granules neither from the point of view of the query processing nor from the point of view of the mechanism used to present the results to the user. Thus, only the GPS locations are usually considered (Sistla et al., 1997; Prabhakar et al., 2002; Mokbel et al., 2005; Cai et al., 2006; Gedik and Liu, 2006; Ding et al., 2008; Ilarri et al., 2010). Moreover, the current approach offers two additional benefits:

1. **Retrieval of the current locations of the objects in an answer.** The locations of the objects in the answer are retrieved/updated as necessary and can be shown to the user using a convenient representation mechanism. On the contrary, as studied
in Ilarri et al. (2010), most proposals do not aim at maintaining the answers up-to-date with the current locations of the objects (a notable exception is Lazaridis et al. (2002)), let alone representing these locations using different mechanisms.

2. Distributed query processing. The current proposal has been integrated in LOQOMOTION (Ilarri et al., 2006a), a system to process continuous location-dependent queries in a distributed environment. Other proposals either consider a centralized architecture (Sistla et al., 1997; Prabhakar et al., 2002; Mokbel et al., 2005; Ding et al., 2008) or a distributed query processing approach on the moving objects themselves (Cai et al., 2006; Gedik and Liu, 2006), which has some disadvantages regarding the possibility to overload the moving objects with query processing tasks (Ilarri et al., 2010). The proposal in Jayaputera and Taniar (2005a,b) also considers a distributed architecture on a fixed infrastructure. However, it can only process range queries over static objects and regarding the location of the mobile user.

Thus, the originality of this work is that it studies in depth the use of location granules in location-dependent queries about moving objects and presents the first approach that is able to process these queries in a distributed and wireless environment, without overloading the mobile devices of the users and with the capability to retrieve (and represent appropriately) the locations of the objects in the answer set. Besides, an extensive experimental evaluation has been performed, that proves the feasibility of processing location-dependent queries with location granules in such an environment, which has not been studied before.

It should be clarified that this work is closer to the field of moving object databases (Gütting and Schneider, 2005; Wolfson et al., 1998), and particularly location-dependent queries (Gedik and Liu, 2006; Ilarri et al., 2010; Prabhakar et al., 2002), than to the general field of spatial databases (Manolopoulos et al., 2005; Rigaux et al., 2002), as it specifically deals with the processing of location-dependent queries about moving objects in distributed and mobile environments. Thus, it does not concern about developing solutions specifically aimed at Geographic Information Systems but solutions more relevant to Location-Based Services (Schiller and Voisard, 2004) and mobile computing (Imielinski and Korth, 1996; Pitoura and Samaras, 1997). However, these two fields are very related and the delimitation between them is not always clear, as acknowledged in the literature\(^{19}\). The work presented in the current paper bridges the gap between both fields by enhancing location-dependent queries in mobile environments with the possibility to express appropriate spatial granularities.

The importance of considering a suitable spatial granularity has been emphasized in previous related work, which reinforces the importance of the proposed solution to manage location-dependent queries with location granules in a mobile environment. Thus, several proposals on spatial databases and geographic information systems deal with the problem of managing spatial data at different levels of detail (e.g., Fonseca et al. (2002); Camossi et al. (2003)). Some of these are concerned about semantic

\(^{19}\)Thus, for example, the work by Gütting and Schneider (2005) focuses on moving objects but also deals with spatial and spatio-temporal databases, and the one by Manolopoulos et al. (2005) deals with spatial databases but also includes some topics about moving objects.
granularity (Fonseca et al., 2002) (which refers to different levels of specification of an object from a semantic point of view, such as “amount of water” vs. “river” or “sea”), while others are concerned about spatial granularity (Camossi et al., 2003) (which implies different levels of spatial resolution/scale), according to the distinction made in Fonseca et al. (2002). Some proposals also consider the time dimension, and therefore the temporal granularities and spatio-temporal granularities (e.g., Belussi et al. (2009)). As opposed to these proposals, the work presented in the current paper considers the granularity of locations (not spatial entities), and focuses on enhancing the expressiveness of the queries instead of dealing with different levels of detail/specification. Moreover, the current proposal mainly concerns continuous location-dependent queries about moving objects in mobile environments. Regarding the modeling of granules, a preliminary approach based on the use of ontologies has also been proposed in Bobed et al. (2010), which can be seen as complementary to these works.

In the authors’ view, the most related work in the context of the current paper focuses on privacy issues (Atallah and Frikken, 2004; Bellavista et al., 2005; Duckham and Kulik, 2005), where deliberately degrading the quality of the location information is considered to achieve a balance between privacy and service quality. The goal of such proposals is to expose the location data only with the precision required for the location-based service considered. On the contrary, the focus in the current paper is on query processing issues and on analyzing how the use of location granules can enhance the expressiveness of location-dependent queries. Work on location uncertainty (Trajcevski et al., 2004) can also be considered related, as the precise geographic location of an object within a location granule is unknown. The difference with the current approach is that the location granules are defined according to the required granularity (and so they are value-added), while location uncertainty is undesirable. Nevertheless, an initial approach to deal with location granules with uncertainty has been proposed in Ilarri et al. (2009).

It is also worth mentioning the location granularity mismatch, which occurs when the granularity of the locations stored in a database and the granularity of the locations specified in a location-dependent query are different. In this case, a location translation (called location leveling) must be performed based on metadata describing a location hierarchy and the relationships between locations at different granularity (Seydim et al., 2001). This problem is orthogonal to the proposal in the current paper, as the focus here is on query processing and granule maps are used to perform the required translations. Related to this, but considering the time dimension instead of the location dimension, the problem of managing different temporal granularities (Dyreson et al., 2000) can be mentioned.

Finally, in Roth (2005) the Nimbus framework is proposed to provide locations with the appropriate granularity. As opposed to the current proposal, Nimbus deals with location granularity at a database level. Therefore, the user is not able to define his/her own granule maps.

8. Conclusions and future work

In the current paper, the importance of considering the appropriate location granularity for the processing of location-dependent queries has been highlighted. Moreover,
a suitable approach to process continuous location-dependent queries with location granules has been presented. The main advantages of the proposal are as follows:

1. **Higher query expressivity.** The use of location granules greatly increases the expressiveness of the location-dependent queries and the range of applications that can benefit from the query processing. This can be seen from a double perspective:
   - *Considering the input.* It allows defining location-dependent queries with the semantics needed (i.e., referring to locations using the required language/terminology).
   - *Considering the output.* The results can be represented in a way that is useful to the user. Different representation mechanisms can be applied independently of the query processing and the location granularity required.

In other words, due to the use of location granules it is possible to define queries that otherwise would not be possible. Moreover, the location granules which are projected as a result of a query can be represented following different approaches that are independent of the query processing itself. These two elements provide a great flexibility and prove the usefulness of location granules.

2. **Distributed query processing.** The proposal is integrated in an existing system for the distributed processing of continuous location-dependent queries in mobile environments (Ilarri et al., 2006a). The distributed approach provides interesting advantages, such as: 1) the mobile devices of the users are not overloaded with query processing and wireless communication tasks, 2) the performance and scalability of the query processing is increased (queries about different geographic areas are executed in parallel), 3) the scalability can be easily increased by adding new proxies, and 4) the service availability can be easily enhanced (by replicating data and functionalities).

3. **Good performance.** The advantages of location granules do not come at the expense of performance. An extensive experimental evaluation has been performed, with both real and synthetic granule maps and under different conditions (number of moving objects, number of granules, etc.), which proves the above mentioned observation. While the management of location granules adds a certain overhead to the query processing (e.g., it requires performing several geometric computations), this cost is very limited and affordable. Moreover, the experiments also show that the proposal achieves a good accuracy. Last but not least, when location granules are used for the presentation of results and an incremental update communication approach is considered, the amount of wireless communications is minimized. Thus, dealing with coarse location granules reduces the number of location updates that must be communicated to the mobile device of the user. Similarly, in the internals of the query processing the effort needed to keep track of the current location of an object is also smaller when location granule maps are specified in the query constraints. The prototype implemented shows the good performance, flexibility, and feasibility of the distributed approach.
The work presented in this paper is the first proposal to process continuous location-dependent queries with location granules in a mobile environment. Thus, whereas other works have highlighted the interest of using different spatial granularities, an in-depth study of the concept of location granule in the context of location-dependent queries about moving objects in distributed and wireless environments was missing. Besides, the proposal presented in this paper is also able to retrieve the locations of the moving objects in an answer, that can then be represented appropriately according to the location granularity required. An extensive experimental evaluation shows the feasibility of the proposal.

Future work includes developing a mechanism to deal with uncertainty issues (a preliminary approach for the case of inside and nearest queries with location granules was presented in Ilarri et al. (2009)). It could also be interesting to study how the use of spatial hierarchies (Malinowski and Zimányi, 2005) or ontologies (Staab and Studer, 2004) can assist in managing and comparing granule maps. For example, using spatial ontologies with a reasoner can help in translating between granule maps without performing geographic computations (e.g., if a person is within a certain room and the room is within a building, then the person is within that building). Regarding this, an initial work about the addition of semantics to location granules has been presented in Bobed et al. (2010). However, the above mentioned issue needs to be studied in more detail.

Acknowledgements

This work was supported by the CICYT project TIN2007-68091-C02-02. The authors thank Antonio Corral for his useful comments. The anonymous reviewers also provided very interesting comments that helped to improve this work.

APPENDICES

Appendix A. Query processing algorithm

In Figure A.1, the main aspects of the basic algorithm needed to process an inside constraint (illustrated in Section 4) are shown. Location granules of a certain type could be considered for the reference object, for the target objects, or for both. So, three main parts can be distinguished in the algorithm, as explained in the rest of this appendix. The complexity of the algorithm presented is analyzed in an electronic appendix (http://webdiis.unizar.es/~silarri/PUBLICATIONS/jss2011/).

Appendix A.1. Part 1: Interpret the location of the reference object with the required granularity (lines 1-14)

In this part of the algorithm, the following steps take place:

- First, the location of the reference object is retrieved (line 1).
- Then, it is checked whether such a location must be interpreted as a location granule (line 2). This is specified as part of the constraint.
Algorithm 1 (main algorithm): processConstraint(c)

**Require:** c is the constraint that must be processed, LocationServer is the location server that stores the information about the objects, baseOfMaps is a collection of available granule maps, and G (shorthand of Geometry) is an object specialized in performing geometric operations.

**Ensure:** Retrieves the objects that satisfy the constraint.

1: coordinatesOfRefObj \(\leftarrow\) LocationServer.queryLocation(c.refObj.id);
2: if (c.refGranuleID \(<>\) null) then
3:  idOfGrMapRef \(\leftarrow\) c.refGranuleMapID;
4:  grMapRef \(\leftarrow\) baseOfMaps.getMap(idOfGrMapRef);
5:  granuleRef \(\leftarrow\) grMapRef.getGranule(idOfGrMapRef);
6:  relevArea \(\leftarrow\) \(G\).extendAreaBy(granuleRef.area, c.relevRadius);
7: else if (c.refObj.granuleMapID \(<>\) null) then
8:  idOfGrMapRef \(\leftarrow\) c.refObj.granuleMapID;
9:  grMapRef \(\leftarrow\) baseOfMaps.getMap(idOfGrMapRef);
10:  granuleForRef \(\leftarrow\) grMapRef.getGranule(coordinatesOfRefObj);
11:  relevArea \(\leftarrow\) \(G\).extendAreaBy(granuleForRef.area, c.relevRadius);
12: else
13:  relevArea \(\leftarrow\) \(G\).getCircle(coordinatesOfRefObj, c.relevRadius); /* Center, radius */
14: end if
15: if (c.targetClass.granuleMapID \(<>\) null) then
16:  idOfGrMapTarget \(\leftarrow\) c.targetClass.granuleMapID;
17:  grMapTarget \(\leftarrow\) baseOfMaps.getMap(idOfGrMapTarget);
18:  intersectGrs \(\leftarrow\) \(G\).getIntersectingGranules(relevArea, grMapTarget); /* The new enlarged relevant area is the union of these granules. However, there is no need to perform the actual union. */
19:  objects \(\leftarrow\) \(\emptyset\);
20:  for all intersectGr \(\in\) intersectGrs do
21:    objects \(\leftarrow\) objects \(\cup\) getObjectsInArea(intersectGr.area);
22:  end for
23: else
24:  if (c.refObj.granuleMapID \(<>\) null) then
25:    objects \(\leftarrow\) getObjectsInArea(relevArea);
26:  else
27:    objects \(\leftarrow\) LocationServer.getObjectsWithin(relevArea);
28:  end if
29: end if
30: return objects;

Figure A.1: Algorithm to process location-dependent constraints with location granules
Algorithm 2 (auxiliary algorithm): getObjectsInArea(a)

Require: \( a \) is an area, \( \text{LocationServer} \) is the location server that stores the information about the objects, and \( G \) (shorthand of \( \text{Geometry} \)) is an object specialized in performing geometric operations.

Ensure: Retrieves the objects within the area specified.

1: \( mbr \leftarrow G.\text{getMBR}(a) \);
2: \( \text{objectsToReturn} \leftarrow \text{LocationServer}.\text{getObjectsWithin}(mbr) \);
3: for all \( \text{objectToCheck} \in \text{objectsToReturn} \) do
4:   \( \text{if } (G.\text{contains}(a, \text{objectToCheck}.\text{coordinates}) = \text{false}) \text{ then} \)
5:   \( \text{objectsToReturn} \leftarrow (\text{objectsToReturn} - \{\text{objectToCheck}\}) \);
6: \( \text{end if} \)
7: \( \text{end for} \)
8: \( \text{return objectsToReturn} \);

Figure A.2: Algorithm to process location-dependent constraints with location granules: auxiliary algorithm

- If so, the identifier of the granule map to consider, also specified in the constraint, is obtained (line 3). This identifier is used to retrieve (from a repository of granule maps) the proper granule map (line 4). The corresponding granule map is used to transform the location retrieved in line 1 (e.g., a GPS location) into the appropriate location granule (line 5).

- Finally, a relevant area is obtained, which represents the interesting geographic area around the reference object. If the location of the reference object must be interpreted as a location granule (line 2), this area is obtained by extending the corresponding location granule according to the relevant radius specified in the constraint (line 6); an example of this computation was shown in Figure 8.a (on page 16).

As mentioned in Section 2.2, the reference of the constraint can also be a static granule (see Figure 1). In this case, the relevant area is obtained by extending such a granule according to the relevant radius (lines 7-11).

If a location granule was not specified for the reference object (line 12), then the relevant area is simply a circle with radius the specified distance and centered on the location of the reference object (line 13).

Appendix A.2. Part 2: Modify the relevant area obtained, if necessary, according to the location granularity required for the target objects (lines 15-18)

In this part of the algorithm, the following steps take place:

- First, it is checked whether the locations of the target objects must be interpreted as location granules (line 15). This is specified as part of the query constraint.
If that is the case, and similar to what was explained above, the identifier of the granule map to consider (line 16) is first used to retrieve (from a repository of granule maps) the granule map that must be used (line 17).

Next, the granules (from those contained in that map) that intersect the relevant area around the reference object (obtained in lines 1-14) are computed, obtaining the relevant granules (line 18). The new enlarged relevant area would be the union of the areas of these granules (an example of this computation was shown in Figure 8.b, on page 16).

Appendix A.3. Part 3: Obtain and return the target objects (lines 19-29)

In this part of the algorithm, the following steps take place:

- If an enlarged relevant area was computed, then the relevant granules in that area are processed iteratively to retrieve the objects within those granules (lines 19-22). For this, the auxiliary Algorithm 2 is used. Algorithm 2 performs three steps to get the objects within an area: 1) the Minimum Bounding Rectangle (MBR) of the corresponding area is computed (line 1), 2) the objects within that area are retrieved by querying the Location Server module (see Section 3.2) in line 2, and 3) the false positives (objects that are within the MBR of the area but not precisely within the area) are removed by checking that each of those objects are actually within the area (lines 3-7).

- If not (line 23), then the objects that are within the relevant area computed before in line 6, 11, or 13 of the algorithm are retrieved. If there is a granule map specified for the reference object (line 24), then the relevant area is not a simple circle and therefore the auxiliary Algorithm 2 is used to retrieve the objects within that area (line 25). Otherwise (line 26), the objects are simply retrieved by asking the Location Server about the objects within the corresponding circle (line 27); no MBRs are needed in this case because the condition that an object is within a circle of radius $r$ can be easily expressed in standard SQL by requiring the object to be at most at distance $r$ of the center of the circle.

If it is assumed that the Location Server is based on a Database Management System (DBMS) that supports queries about objects within areas with arbitrary shape (i.e., a database engine with spatial capabilities), then the use of MBRs and the subsequent refinement step can be avoided (i.e., the auxiliary Algorithm 2 would be unnecessary) and the objects within the interesting area can be retrieved directly.\(^\text{20}\)

References


\(^{20}\text{For example, Oracle Spatial (http://www.oracle.com/technology/products/spatial/index.html, accessed March 5, 2011) provides a spatial operator SDO\_RELATE that, used with a mask CONTAINS, facilitates finding objects within a given area.}\)


Ilarri, S., Mena, E., Illarramendi, A., 2010. Location-dependent query processing: Where we are and where we are heading. ACM Comput. Surv. 42 (3), 1–73.


