Processing Location-Dependent Queries with Location Granules[⋆]

Sergio Ilarri, Eduardo Mena, and Carlos Bobed

IIS Department, University of Zaragoza
Maria de Luna 1, 50018, Zaragoza, Spain
{silarri,emena,cbobed}@unizar.es

Abstract. Existing approaches for the processing of location-dependent queries implicitly assume location data expressed at maximum precision (e.g., GPS). However, there exist applications where managing location data with this precision is not required and may even be inconvenient. Thus, for example, a user could just be interested in the city that a train is currently traversing. In this situation, retrieving the precise geographic coordinates would lead to an unnecessary overhead in terms of the data communications needed to track the continuously changing current location. Moreover, the user would need some mechanism to translate the coordinates into the corresponding city.

In this paper, we stress the importance of a query processing approach that adapts itself to the needs of the user and the level of resolution required: the user should be able to express queries and retrieve results according to his/her own terminology for locations (GPS, cities, states, provinces, or any other geographical area). We have implemented a prototype to test the functionality and the interest of location granules and show the independence between the query processing approach and different ways of presenting the answers.

1 Introduction

Nowadays, there is a great interest in mobile computing, motivated by an ever-increasing use of mobile devices, that aims at providing data access anywhere and at anytime. Recently, there has been an intensive research effort in *location-based services* (LBS) [1]. These services provide value-added by considering the locations of the mobile users in order to offer more customized information.

How to efficiently process a continuous location-dependent query (a query whose answer depends on the locations of objects and is refreshed automatically) is one of the greatest challenges in location-based services. Thus, these queries require a continuous monitoring of the locations of relevant moving objects in order to keep the answer up-to-date efficiently. For example, a user with a PDA (*Personal Digital Assistant*) may want to locate available taxi cabs that are near him/her while he/she is walking home on a rainy day. The answer to this query must be continuously refreshed because it can change immediately, as the

[⋆] This work is supported by the CICYT project TIN2004-07999-C02-02.

user and the taxi cabs move independently. Moreover, even if the set of taxis satisfying the query condition does not change, their locations and distances to the user do change continuously, and 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 that is shown to the user).

Existing works on location-dependent query processing implicitly assume GPS locations for the objects in a scenario [2,3]. 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 could just need to consider in which city the train currently is, and its exact coordinates may be irrelevant. For such applications, we consider useful to define the concept of *location granule* as a set of physical locations. In our previous example, every city would be a location granule. Other examples of location granules could be: freeways, buildings, offices in a building, etc. The idea is that the user should be able to express queries and retrieve results according to the idea of "location" that he/she requires, whether he/she needs to talk in terms of GPS locations or locations at a different *resolution*. The use of location granules can have an impact on:

– *The presentation of results.* The user may want to retrieve the 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 objects are) because it is more appropriate for his/her context. Notice that, in some cases, he/she may even be unable to interpret an answer in terms of GPS locations; for example, if he/she only knows about "cities", then returning GPS locations is useless because he/she does not know how to obtain which city corresponds with a GPS location. Independently of the required location granularity, the answer can be presented to the user using different mechanisms (e.g., different types of graphical, sound-based, textual, etc., representations).
– *The semantics of the queries.* Location-dependent constraints can be interpreted in terms of location granules. As an example, the user may be interested in the cars that are near the city where another car is currently present. Notice that the user may have no idea about geographic coordinates or the boundaries of the cities.
– *The performance of the query processing.* In a general case, the continuous query processing will demand less resources when coarse location granules are used. For example, if a mobile user is interested in the GPS locations of certain cars, the location data of the relevant cars should be communicated very frequently to the mobile device (the GPS locations of the cars are continuously changing). However, less wireless communications are required if he/she is just interested in the part of the city (e.g., north, south, west, east, center) where the cars are.

In this paper, we prove the utility of dealing with locations expressed at different levels of granularity when processing location-dependent queries. In Sect. 2

we define the architecture and concepts that will be used along the paper. In Sect. 3, we justify the importance of location granules and how they can be used to express queries with the semantics that is useful for the user. In Sect. 4, we present some related works. Finally, conclusions and future work appear in Sect. 5.

2 Location Granules and Architecture

We focus on the processing of location-dependent queries with location granules, where we consider the following three datatypes: *Object*, *Location Granule*, and *Location Granule Mapping*.

Definition 1. ***Datatype Object***
We call object to any moving entity (e.g., a person or a car) of interest. A value of type *Object* (in the following, denoted by O) is defined by a tuple with the structure <**id, loc, class**>, where *id* is the identifier of a moving object, *loc* is the GPS location of the object, and *class* indicates the type of object (e.g., *vehicle*); besides, there may be other attributes specific to every object class (e.g., an object of class *vehicle* may have an attribute *maxSpeed*). A **showObject** operation is also defined for an object, which represents the object (by using graphics, sounds, text, etc.).

Definition 2. ***Datatype Location Granule***
A location granule is a geographical area (not necessarily contiguous). A value of type *Location Granule* (in the following, denoted by G) is defined by a tuple <**id, Fs**>, where *id* is the granule identifier and *Fs* is a set of bidimensional figures $Fs = \{F1,\ F2,\ ...,\ Fn\}$ (e.g., polygons, circles, etc.). The following operations are defined for location granules:

- **inGranule: G x GPS → boolean**, with *GPS* = set of possible GPS locations
  inGranule(g, loc) = true ⇔ ∃ *Fi* ∈ *g.Fs* | *contains(Fi, loc)*
  contains(Fi, loc) = true ⇔ *loc* is within *Fi*
- **distanceCentroidGranule: G x GPS → real**
  distanceCentroidGranule(g, loc) = min{*distance(loc, centroid(Fi))*, ∀ *Fi* ∈ *g.Fs*}
- **distanceLimitsGranule: G x GPS → real**
  distanceLimitsGranule(g, loc) = min{*distance(loc, Fi)*, ∀ *Fi* ∈ *g.Fs*}

Besides, a **showGranule** operation is defined for a granule, which shows the location granule to the user by using graphics, sounds, text, etc.

Definition 3. ***Datatype Location Granule Mapping***
A location granule mapping is a data structure that, given a GPS location, allows to obtain one or more location granules containing that GPS location. A value of type *Location Granule Mapping* (in the following, denoted by M) is defined by a tuple with the structure <**id, granules**> where *id* is the granule mapping identifier and *granules* is the set of location granules a certain GPS location can be translated to using that mapping. The following operations are defined for location granule mappings:

- **getGranules: M x GPS → {G}**
  getGranules(m, loc) $= \{g \mid g \in$ *m.granules* $\wedge$ *inGranule(g, loc)*$\}$
- **getNearestGranule: M x GPS x D → G**
  with $D = \{$*distanceCentroidGranule, distanceLimitsGranule*$\}$ (see Def. 2)
  getNearestGranule(m, loc, d) $= g \mid g \in$ *m.granules* $\wedge$ *inGranule(g, loc)* $\wedge$
  $\neg\exists$ *g' $\in$ m.granules | inGranule(g', loc) $\wedge$ d(g', loc) $<$ d(g, loc)*
- **getGranulesOfObject: M x O → {G}**
  getGranulesOfObject(m, o) = getGranules(m, o.loc)
- **getNearestGranuleOfObject: M x O x D → G**
  getNearestGranuleOfObject(m, o, d) = getNearestGranule(m, o.loc, d)

The basic architecture that we propose for the processing of continuous location-dependent queries with location granules is shown in Fig. 1. A *Mobile User* launches queries using his/her PDA, which are processed by a *Query Processor* executing on a *Server*. The user can reference his/her own location granule mappings (*User Mappings*) in the queries (User Mappings in use by current queries are stored on a *Mapping Cache* by the Server) or use predefined mappings (*Server Mappings*). The Query Processor interacts with a *Location Server*, which handles location data about moving objects and is able to answer standard SQL-like queries about them (e.g., it could be a moving objects database [4]). Finally, a *Query Table* is used by the Query Processor to store the active queries: changes in the answers to such queries must be communicated to the corresponding mobile devices.

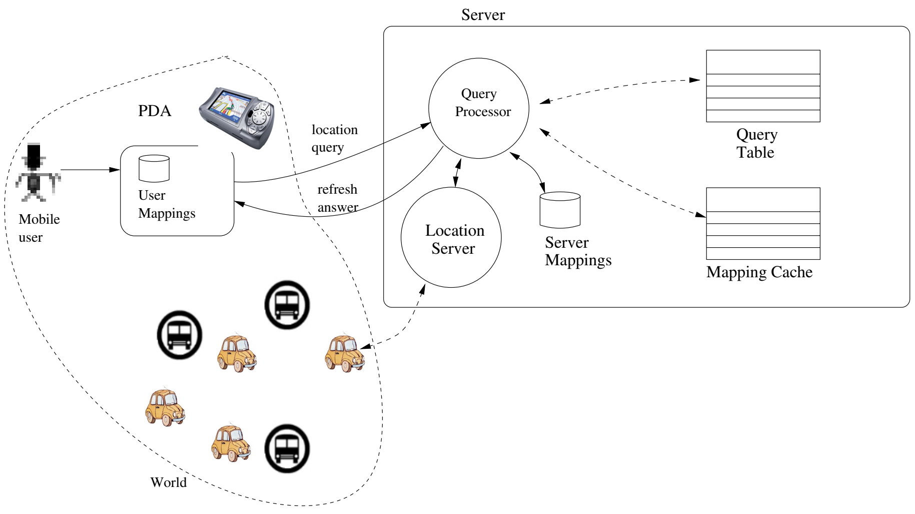


Fig. 1. General architecture for location query processing using location granules

3 Use of Location Granules

In this section, we will use an SQL-like syntax to express the queries; although we could adopt an existing query language (e.g., *SQL/MM Spatial* [5] or FTL [4]), this syntax allows us to emphasize the use of location granules and state our queries concisely.

Specifications of location granules can appear in the SELECT and/or in the WHERE clause of a query, depending on whether the location granules must be used for visualization of results and/or processing of constraints, respectively. A GPS location will be considered for an object if a location granule is not specified instead. In the following, we first explain how location granules allow the user to specify the way the results must be presented. Then, we show that location granules can be also used to specify queries with the required semantics. Of course, both usages can appear simultaneously in the same query. For simplicity, we will use *getNearestGranuleOfObject* (instead of the more general *getGranulesOfObject*, see Sect. 2), which for the sake of brevity will be denoted by *gr* in the queries, and omit the parameter *d* that indicates the distance function that must be used.

3.1 Granules in Query Projections

The user can decide to retrieve location granules as part of the query projections. For example, "SELECT gr(bus25, region) FROM Bus" retrieves the location granule of type *region* corresponding to the current location of *bus25*. When a location granule is retrieved as part of a query answer, it is automatically shown to the user by executing the implemented *showGranule* operation (see Sect. 2). Thus, in the previous sample query, the retrieved granule may be represented, for example, by highlighting the current region on a map (e.g., painting it in black).

Different types of representations (graphical, sound-based, etc.) are possible (some examples are shown in an interactive demonstration applet at `http://sid.cps.unizar.es/ANTARCTICA/LDQP/granulesRepresentation.html`). For example, the granules of moving objects in a scenario such as the one in Figure 2 can be shown to the user in different ways (see Figure 3). In Figure 3.a every region in France is a location granule and it is represented with a different color depending on the number of buses within the granule. Alternatively, in Figure 3.b the granules are also regions of France but they are represented differently: circles in a bus route such that the identifier of each bus appears next to the circle corresponding to the region where the bus is currently.

3.2 Granules in Query Constraints

There are different types of location-dependent constraints [6]. For clarity, we focus on *inside* constraints, which have the general structure *inside(r, obj, target)*, where *r* is called the *relevant radius*, *obj* is the *reference object* of the constraint, and *target* is called the *target class*. Such a constraint would retrieve the objects of the class *target* which are within distance *r* of the object *obj*. Moreover, we can specify location granules in an *inside* constraint instead of *obj*, *target*, or both. We describe in the following these three cases (an interactive demonstration of how these cases are processed is available as a Java applet at `http://sid.cps.unizar.es/ANTARCTICA/LDQP/granules.html`).

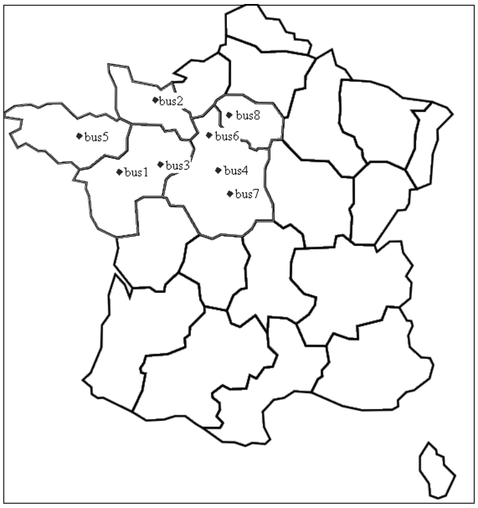
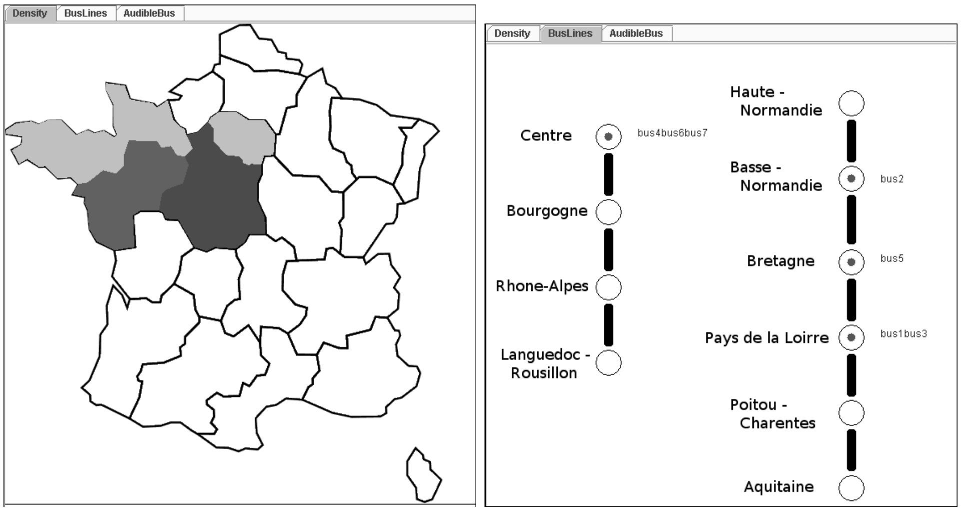


Fig. 2. Map of France with some buses



(a) (b)

Fig. 3. Different presentation mechanisms: (a) painting the regions with a different color depending on the number of buses within, and (b) highlighting the current region on the bus routes

1) Referencing the Granule of a Reference Object. In this case, the *inside* constraint retrieves the *target objects* (objects of the target class) whose distance from the granule of the reference object is not greater than the relevant radius:

$$\textit{inside(r, gr(m, obj), target)} = \{oi \mid oi \in \textit{target} \wedge \ \exists\, p \in GPS \mid \\ \textit{inGranule}(gr(m, obj), p) \wedge \textit{distance}(p, oi.loc) \leq r\}$$

For example, the query "SELECT Car.id FROM Car WHERE inside(130 miles, gr(province, car38), Car)" retrieves the identifiers of the cars within 130 miles of the province where *car38* (the reference object of the constraint) is located.

To obtain the objects that satisfy this type of *inside* constraint, the following operations are performed (see Fig. 4 for an example, where we consider the map of Spain divided in provinces): 1) the granule of the reference object (in the example, the province of *car38*) is obtained (see Fig. 4.a); 2) the area/s corresponding to such a granule is/are enlarged in order to obtain the *relevant area/s* according to the radius relevant to the query (in the example, 130 miles, see Fig. 4.b); and 3) the target objects within that area are retrieved (see Fig. 4.c).

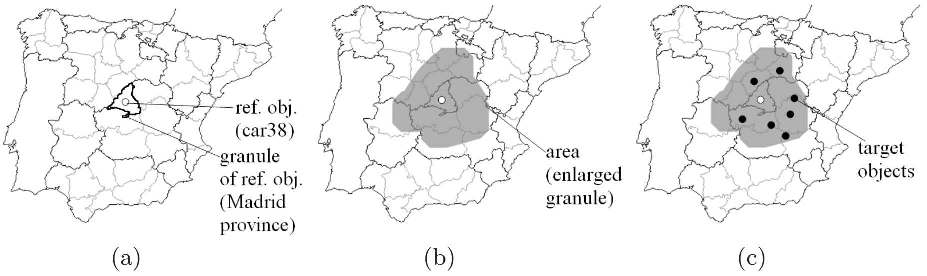


Fig. 4. Granules for the reference object: (a) obtaining the granule of the reference object, (b) obtaining the relevant area, and (c) retrieving the objects within the area

The operation corresponding to the second step, which implies computing the Minkowski sum of the area/s composing the granule and a disk with radius the relevant radius, is called *buffering* in the context of Geographic Information Systems [7].

2) Referencing the Granules of a Target Class. In this case, the *inside* constraint retrieves the target objects located in location granules whose boundaries intersect with a circle of the relevant radius centered on the reference object:

$$\textit{inside(r, obj, gr(m, target))} = \{oi \mid oi \in \textit{target} \wedge \ \exists\, p \in GPS \mid \\ \textit{inGranule}(gr(m, oi), p) \wedge \textit{distance}(p, obj.loc) \leq r\}$$

For example, the query "SELECT Car.id FROM Car WHERE inside(130 miles, car38, gr(province, Car))" retrieves the cars located in provinces whose boundaries are (totally or partially) within 130 miles of *car38.*

To obtain the objects that satisfy this type of *inside* constraint, the following operations are performed: 1) a circular area with the relevant radius (in our example, 130 miles) centered on the current GPS location of the reference object (in the example, *car38*) is computed (see Fig. 5.a); 2) the granules intersected by such an area are determined (see Fig. 5.b); and 3) the target objects within any of those granules are obtained (see Fig. 5.c).

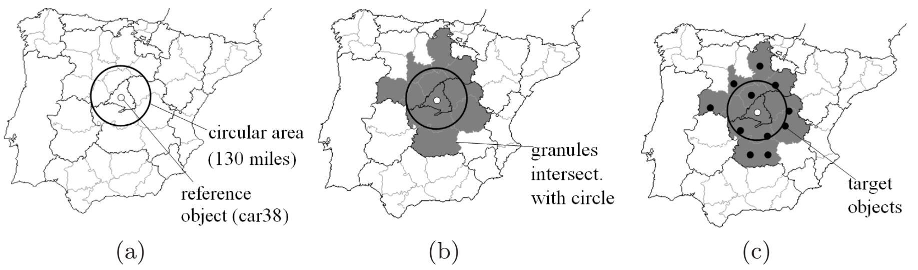


Fig. 5. Granules for the target class: (a) obtaining the circular area, (b) obtaining the granules that intersect, and (c) retrieving the objects within the granules

3) Referencing the Granules of the Reference Object and the Target Class. In this case, the *inside* constraint retrieves the target objects located in location granules whose boundaries are within the relevant radius from the granule of the reference object:

$$\mathit{inside(r,\ gr(m1,\ obj),\ gr(m2,\ target))} = \{oi \mid oi \in \mathit{target} \wedge\ \exists\ p1 \in GPS\ \wedge \\ \exists\ p2 \in GPS \mid\ \mathit{inGranule}(gr(m2, oi), p1) \wedge \mathit{inGranule}(gr(m1, obj), p2) \wedge \\ \mathit{distance}(p1, p2) \leq r\}$$

For example, the query “SELECT Car.id FROM Car WHERE inside(130 miles, gr(province, car38), gr(province, Car))” retrieves the cars located in provinces whose borders are (total or partially) within 130 miles of the province where *car38* is located.

In this case, to obtain an answer, the granule of the reference object is obtained and the area/s corresponding to such a granule is/are enlarged in order to obtain the *relevant area/s* according to the radius relevant to the query (as in case 1, see Fig. 6.a). Then, the set of granules intersected by such an area are determined (as in case 2, see Fig. 6.b). Finally, the target objects within those granules are retrieved (see Fig. 6.c).

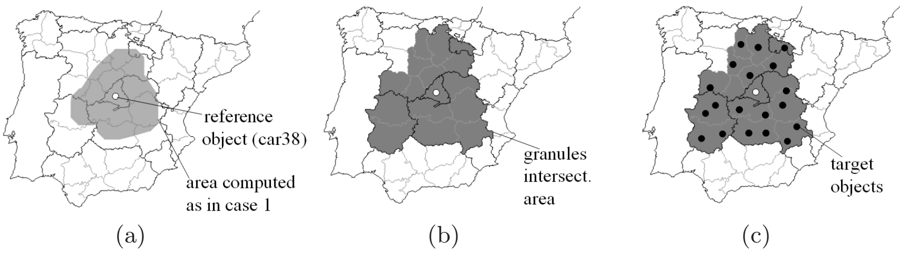


Fig. 6. Granules for the reference object and the target class: (a) obtaining the relevant area, (b) obtaining the granules that intersect it, and (c) retrieving the objects inside

4 Related Work

The most related works in the context of this paper focus on privacy issues [8], where they propose degrading deliberately the quality of location data in order to achieve a balance between privacy and service quality. However, they do not consider means to help to express queries which are relevant to the user.

Several works on spatial databases and geographic information systems deal with the problem of managing spatial data at different levels of detail (e.g., [9,10]). These works focus on the problem of dealing with different levels of detail/specification of the spatial entities, and therefore they do not consider the use of locations at different granularities to enhance the expressiveness of queries.

It is also worth mentioning the so-called *location granularity mismatch*, which occurs when the granularity of the locations stored on a database and the granularity of the locations specified in a location-dependent query are different [11]. This problem is orthogonal to our work, as we focus on query processing and rely on granule mappings to perform the required translations.

Finally, we would like to mention the work in [12], where the *Nimbus* framework is proposed to provide locations with the appropriate granularity. As opposed to ours, it deals with location granularity at the database level; therefore, for example, the user is not able to define his/her own granule mappings.

5 Conclusions and Future Work

In this paper, we have proposed an approach to consider the appropriate location granularity for the processing of location-dependent queries. The use of location granules greatly increases the expressiveness of location-dependent queries and the range of applications that can benefit from the query processing. On the one hand, it allows the user to specify the queries with the needed semantics (i.e., he/she can talk about locations "on his/her own terms"). On the other hand, the results can be represented in a way that is useful for the user; different representation mechanisms can be applied independently of the query processing. Moreover, the performance of the query processing also improves when location granules are used for the presentation of results (e.g., saving wireless communications).

We have also implemented a prototype that shows the flexibility and feasibility of our approach. We have integrated our prototype into an existing GPS-based location-dependent query processing system [6], adding the capability to manage location granules. As future work, we plan to study how the use of spatial hierarchies [13] or ontologies [14] can help us, in certain situations, to manage and compare granule mappings. As another interesting line of future work, we think that our system could also be easily extended to deal with imprecise locations [15] (e.g., cells in a cellular network), as these could be considered just as a special kind of location granules.

References

1. Schiller, J., Voisard, A. (eds.): Location-Based Services. Morgan Kaufmann, San Francisco (2004)
2. Cai, Y., Hua, K.A., Cao, G., Xu, T.: Real-time processing of range-monitoring queries in heterogeneous mobile databases. IEEE TMC 5(7), 931–942 (2006)
3. Gedik, B., Liu, L.: MobiEyes: A distributed location monitoring service using moving location queries. IEEE TMC 5(10), 1384–1402 (2006)
4. Sistla, A.P., Wolfson, O., Chamberlain, S., Dao, S.: Modeling and querying moving objects. In: 13th Intl. Conference on Data Engineering (ICDE 1997), Birmingham, United Kingdom, pp. 422–432. IEEE Computer Society Press, Los Alamitos (1997)
5. Stolze, K.: SQL/MM spatial - the standard to manage spatial data in a relational database system. In: Datenbanksysteme für Business, Technologie und Web (BTW 2003), February 2003. Lecture Notes in Informatics (LNI), vol. 26, pp. 247–264 (2003)
6. Ilarri, S., Mena, E., Illarramendi, A.: Location-dependent queries in mobile contexts: Distributed processing using mobile agents. IEEE TMC 5(8), 1029–1043 (2006)
7. Kreveld, M.: Computational geometry: Its objectives and relation to GIS. Nederlandse Commissie voor Geodesie (NCG), 1–8 (2006)
8. Duckham, M., Kulik, L.: A formal model of obfuscation and negotiation for location privacy. In: Gellersen, H.-W., Want, R., Schmidt, A. (eds.) PERVASIVE 2005. LNCS, vol. 3468, pp. 152–170. Springer, Heidelberg (2005)
9. Fonseca, F., Egenhofer, M., Davis, C., Câmara, G.: Semantic granularity in ontology-driven geographic information systems. Annals of Mathematics and Artificial Intelligence 36(1-2), 121–151 (2002)
10. Camossi, E., Bertolotto, M., Bertino, E., Guerrini, G.: Issues on modeling spatial granularities. In: Kuhn, W., Worboys, M.F., Timpf, S. (eds.) COSIT 2003. LNCS, vol. 2825, Springer, Heidelberg (2003)
11. Seydim, A.Y., Dunham, M.H., Kumar, V.: Location dependent query processing. In: 2nd ACM Intl. Workshop on Data engineering for Wireless and Mobile Access (MobiDe 2001), pp. 47–53. ACM Press, New York (2001)
12. Roth, J.: The role of semantic locations for mobile information access. In: 35th Annual GI Conference, Bonn, Germany, September 2005. Lecture Notes in Informatics (LNI), vol. 2, pp. 538–542 (2005)
13. Malinowski, E., Zimányi, E.: Spatial hierarchies and topological relationships in the spatial MultiDimER model. In: Jackson, M., Nelson, D., Stirk, S. (eds.) Database: Enterprise, Skills and Innovation. LNCS, vol. 3567, pp. 17–28. Springer, Heidelberg (2005)
14. Staab, S., Studer, R. (eds.): Handbook on Ontologies. Intl. Handbooks on Information Systems. Springer, Heidelberg (2004)
15. Trajcevski, G., Wolfson, O., Hinrichs, K., Chamberlain, S.: Managing uncertainty in moving objects databases. ACM TODS 29(3), 463–507 (2004)