Processing Continuous Range Queries on Mobile Objects in Location-Based Services

Dragan H. Stojanovic and Slobodanka J. Djordjevic-Kajan

Faculty of Electronic Engineering, University of Nis
Beogradska 14, 18000 Nis, Serbia and Montenegro
Email: dragans@elfak.ni.ac.yu *

Abstract. In this paper we address the problem of processing continuous range queries on mobile objects. In contrast to regular queries that are evaluated once, a continuous query remains active over period of time and has to be continuously evaluated during this time to provide up to date answers. The continuous range query processing we base on the mobile object data management framework, ARGONAUT, which represents an implementation of the object-oriented data model and the query language for mobile objects. For the purpose of query processing, we assume an available 2D indexing scheme in the underlying DBMS and focus on the refinement stage of the processing. We propose the pre-refinement step, as well as the data structures and the query classes for efficient reevaluation of range queries.

1 Introduction

Advances in wireless communication technologies, Internet-enabled mobile devices and mobile positioning, have given rise to a new class of location-based applications and services. Location-based services (LBS) deliver geographic information and geo-processing power to the mobile/static users in accordance with their current location and preferences, or locations of the static/mobile objects of their interests. LBS are specialized, multi-tiered, component Web GIS applications, which can be published, located and invoked across the wired/wireless Web [7]. Such services, like automatic vehicle location, fleet management, tourist services, transport management, traffic control and digital battlefield, are all based on mobile objects and the management of their continuously changing locations. Mobile objects can report their locations to the LBS server through a wireless interface, or their locations can be obtained through the ground-based radars or satellites. LBS applications require database and application support to model and manage mobile objects in both database and application domain and to support querying on the motion properties of the objects [8, 9]. The main problem and challenge in location-based services development is handling different types of queries on mobile objects. The most prevalent types of location-dependent queries in LBS are range and k-nearest neighbor queries.

* This research was supported in part by Ministry of Science, Technology and Development, Republic of Serbia, and Municipality of Niš, under the project "Geographic Information System for Local Authorities based on Internet/WWW Technologies", Contract No. IT.1.23.0249A.

In this paper we address the problem of processing continuous range queries on mobile objects. In such queries the range can represent the user-selected area, the map window, the polygonal feature, or the area specified by the distance from a point of interest. In contrast to regular queries that are evaluated only once, a continuous query remains active over a period of time and has to be continuously evaluated during this time to provide up to date answers. Such queries may also represent triggers that enable event notification to the registered users. At any time there will be several continuous queries running at the server. Each of this queries needs to be re-evaluated as the objects move. A major challenge for this problem is repeatedly processing all queries within reasonable amount of time as the numbers of objects being tracked and continuous queries increases.

2 Mobile Objects in LBS

LBS are multi-tiered, Internet GIS applications whose architecture is presented in Fig 1. The location of the mobile devices can be determined by using GPS or mobile network triangulation. Such devices can report their locations to the LBS server through a wireless interface, or their locations can be obtained through ground-based radars or satellites. At the LBS server, the data is processed and services, based on such data, are provided to the users. Mobile users may also represent mobile objects of interest to other users of the same or other LBS. Mobile objects in LBS are characterized by point geometry and always move along a path in a network [1]. Vehicles, trains, boats and passengers move following a particular network (roads, railways, rivers, pedestrian tracks). Air traffic also follows a (3-D) network of air corridors. Mobile objects often use fast and/or shortest paths to their destination, depending on the cost criteria (time and/or distance).

The application scenario for LBS that involves mobile objects both as the users of the service and as the tracked objects is as follows [8]. The mobile object registers for the certain location-based service connecting to the LBS server by sending the starting location (coordinates of the starting point or address), starting time, ending location and eventually the set of points of interest that it is going to visit along its route and duration of the stay at each stop. The road network database accessed and managed by the LBS information server stores a network which is the layer supporting the objects trajectories. Such road network database contains geometry for each road segment, attributes used for geocoding, as well as attributes needed for computing travel time and travel distance for that road segment, such as average speed or typical drive time [13]. Road network database can be updated by using real-time traffic information about accidents, roadwork, traffic congestions, etc [12].

Given the starting motion parameters, LBS server calculates the shortest cost (distance or travel-time) path in a road network and expected trajectory of the mobile object [13]. The expected trajectory is stored in the mobile object database and retrieved and stored in a LBS client on the mobile computer. The LBS server also sends to the client an uncertainty threshold value. The trajectory of the mobile object is a polyline in three-dimensional space (two-dimensional space and time) represented as a sequence of points (x_i, y_i, t_i). Such trajectory is only an approximation because the

object does not move in straight lines at constant speed. The number of points along the trajectory is proportional to the accuracy of such approximation. An additional parameter mt_i for every point defines the type of motion during period $[t_i, t_{i+1}]$. The three motion types are defined: *punctual* - the mobile object isn't tracked and its location isn't defined during certain time period; *stepwise* - the mobile object does not move during the time period; *linear* - the mobile object move along the straight line from (x_i, y_i) to (x_{i+1}, y_{i+1}), and at constant speed. When a mobile object moves along the curve segments with variable speed, the fourth motion type, *interpolated,* must be defined to represents the motion by an interpolation function.

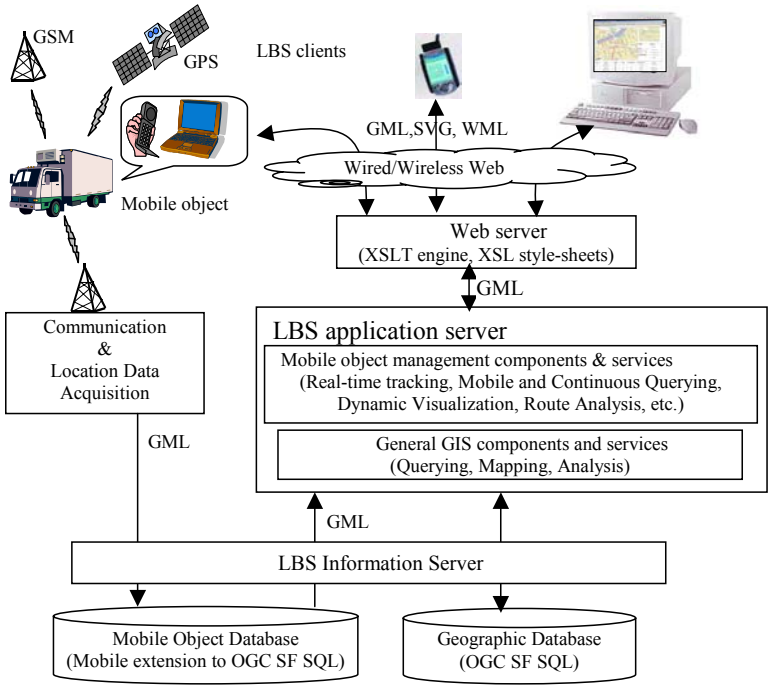

Fig. 1. The LBS Architecture

Based on the expected trajectory of the mobile object, at any point in time t (past or future) between the start and end times of the trip, both the LBS server and the LBS client can compute the expected location of the mobile object at time t. The uncertainty threshold specifies the responsibility of the mobile object to send the location update to the LBS server if its current location is deviated from its expected location by defined uncertainty threshold. With every location update, the expected trajectory of the mobile object from that location to the destination must be updated by $\Delta t = \pm ut / v$, where *ut* is uncertainty threshold and *v* is the average speed for the current road segment. To enable prediction of a mobile object location in the (near) future when the travel attributes aren't available in a road network database, with every location update the mobile object must send to LBS server its current speed. Again, the mobile object has to send the location update to the LBS server if its current location is devi-

ated from its expected location by defined uncertainty threshold. When the route of the mobile object isn't known a priori, because of, for example, the unknown destination, besides the location and the speed, one additional attribute, the direction, must be specified. It enables LBS server to associate the mobile object to the road segment of the underlying network, and define and store a new part of the object's trajectory from the last registered location, according to the shortest cost path between these two locations.

3 Modeling and Querying Mobile Objects in the Argonaut Framework

For the purpose of modeling and management of mobile object location data we have developed the standard-based component framework, named Argonaut [9]. The foundational data model of the Argonaut framework was developed to support conceptual modeling and querying of discretely and continuously changeable properties of geographic features. The Argonaut data model is extensible, object-oriented, and specified using UML class diagram notation. We base our modeling approach on the comprehensive framework of data types and rich algebra of operators defined in [3], but extending their approach to the object-oriented modeling paradigm, and representation of the past, current, as well as the future motion of the mobile objects, moving on the transportation network [1, 8].

The ARGONAUT data model is based on the extension of the OGC Simple Features model, also adopted by ISO TC 211 [4]. The standard defines the abstract *Geometry* class, and its hierarchy of the specialized geometric classes (*Point*, *LineString*, *Polygon*, *MultiPoint*, etc.). The attributes and operations defined within the geometry classes support specification of topological relations, direction relations, metric operations and operations that support spatial analysis (point-set operations) on appropriate geometry types. The time dimension of the mobile object is specified through the *TimeObject* class hierarchy, defined in accordance with ISO TC 211 Temporal Schema [5] (*TimeInstant*, *TimePeriod TimeDuration*, *MultiTimeInstant* etc.). The *TimeObject* class includes the attributes and the operations for specifying topological relations, metric operations and point set operations on time dimension.

The base class for introducing mobility of features and continuous change of their geometric properties is an abstract *MobileGeometry* class, as the root of the extensible hierarchy of classes for specifying mobile geometries (Fig 2). For every class in the *Geometry* class hierarchy an appropriate class for the representation of the mobile geometry is defined (*MobilePoint*, *MobileLineString*, *MobilePolygon*, *MobileMultiPoint*, etc.). Any of these classes appropriately restrict the *Geometry* class aggregated within the *MotionSlice* class (<<restriction>> stereotype). Being a specialization of the *Geometry* class, a *MobileGeometry* and its specialized classes can be treated in the same way as any other geometric object, i.e. it can represent the geometric property of any feature which is dynamic in nature and participate in all geometric operations and relations. An instance of the *MotionSlice* class, aggregated by the *MobileGeometry* class with multiplicity 0..n, represents the registered location of a mobile geometry, by containing a geometry value (instance of *Geometry* class), the valid time of such

value (instance of the *TimeInstant* class) and the motion type (value of enumeration type *MotionType*), which describes the way geometry changes between two successively registered geometries. The ARGONAUT data model provides four enumerated values for motion types, and those are: *Punctual*, *Stepwise*, *Linear* and *Interpolated*. The first three motion types don't require any additional information to be included in the data model, but for the *Interpolated* motion type, the reference to the *Interpolation* class, with defined interpolation parameters, must be specified. The motion of the mobile object also causes continuous change of its non-spatial properties like distance from some static or a mobile object or topological relation between two mobile objects, as elaborated in [2]. Using the same approach, the ARGONAUT data model defines *MobileBoolean* and *MobileDouble* classes, by inheritance from *Boolean* and *Double* base classes respectively and aggregation of appropriate *MotionBoolSlices* or *MotionDoubleSlices* classes.

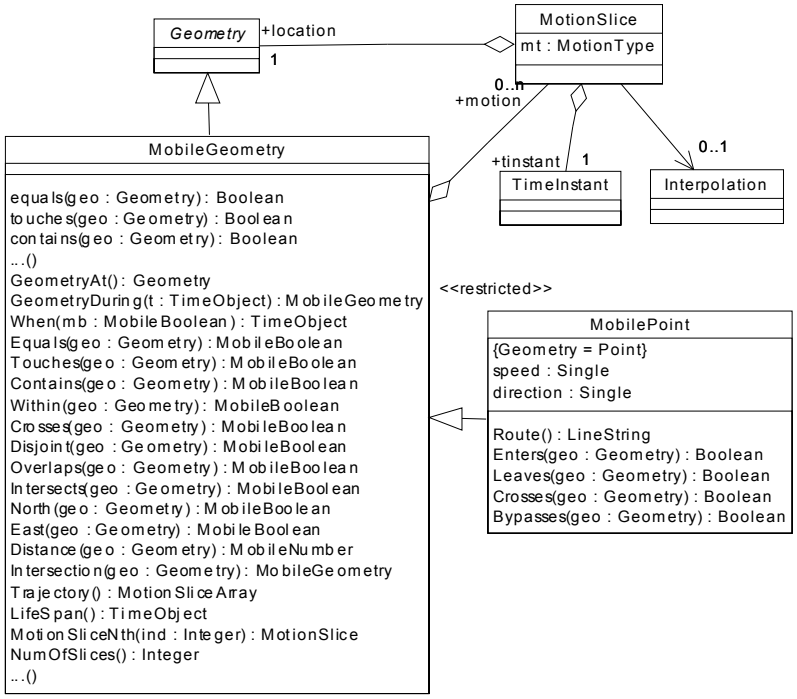

Fig. 2. The foundation of the ARGONAUT data model

The ARGONAUT data model overrides the topological relations, the direction relations, the metric operations and the spatial analysis operations inherited from the base *Geometry* class [4]. These relations and operations take the mobile or non-mobile geometry argument and generate non-mobile results (Boolean, Double, Geometry, etc.). Specialized topological and direction relations, like *touches* and *contains*, have single argument of the *Geometry* type (which could also be the *MobileGeometry*) and return the Boolean *true* value indicating that such relations are satisfied during the

lifespan of the *MobileGeometry* argument(s) according to temporal aggregation defined in [2]. Such operations correspond to spatio-temporal predicates. Metric and spatial analysis overridden operations can not be considered as predicates, and for the *MobileGeometry* argument operation is delegated by default to the geometry value at the current time instant defined using *GeometryAt(Now)* operation. The ARGONAUT data model also defines mobile variants of mentioned non-mobile relations and operations, named with starting capital letter in Fig 2. Such operations correspond to the temporally lifted operations that handle non-mobile or mobile geometry arguments and generate mobile result, since for mobile argument(s) different results are generated in the course of time. The motion type associated with the mobile result depends on the operator or relation. In general, it is not sufficient to inherit the motion type of the mobile argument. For example, for mobile points characterized by the linear motion type, the quadratic interpolation method is needed to represent the mobile result of the *distance* operation.

To support the querying of mobile objects following operations are defined. The *Lifespan* operation returns time object defining the whole life span of the mobile object. The aforementioned *GeometryAt* operation returns geometry value at specific time instant. In order to restrict the sequence of motion slices of a mobile object according to specific time object, *GeometryDuring* operation is defined. That operation restricts the mobile object to only those motion slices from the sequence that belong to the specified time object given as an argument. The *When* operation returns the time object during which the mobile object satisfies the criteria specified by a mobile Boolean argument. The *Route* operation returns *Geometry* result representing the path traversed by the mobile geometry. All these operations are overridden in specific mobile classes inherited from the *MobileGeometry* class. To support manipulation of mobile properties of type *MobileBoolean* and *MobileDouble*, predicates and operations (*and*, *not*, *max*, *add*, *lt*, etc) and their temporally lifted counterparts (*And*, *Not*, *Max*, *Add*, *Lt*, etc.), are defined accordingly [9].

Modeling mobile point objects that move continuously over a predefined network infrastructure, as existed in LBS, are provided by the *MobilePoint* class (Fig 2). The *MobilePoint* class inherits the *MobileGeometry* class and thus inherits and overrides aforementioned spatio-temporal operations and relations. Two attributes, *speed* and *direction*, are defined when road network data don't contain travel attributes (average speed, travel time, etc), or the route of the mobile object is not predefined (e.g. the destination isn't known beforehand). The *Route* operation defined in *MobileGeometry* class is redefined in inherited classes, because for different mobile geometry types, the trajectory operation returns specific geometric result. Such operation applied to mobile line string object yields *Polygon* object as result, and for mobile point object with linear motion type returns a *LineString* geometric object.

The model defines relations arisen from the motion of mobile point over mobile or static polygonal area, such as *Enters*, *Leaves*, *Crosses*, *Bypasses*, etc. as proposed in [2] (Fig 3). We define those relations for the mobile/static point and the polyline objects using the same semantics and definitions given in [2] for the mobile point and the polygon. Those operations are particularly useful in querying mobile points moving on the network paths. Such relations can be defined by the successive application of the several basic mobile relations and enable examination of changing spatial relations over time by simply sequencing basic spatio-temporal predicates. Thus mobile

object enters in the area of the static or mobile polygonal object during the given time period, if it was outside of the polygon at the beginning of the period (*Disjoint* relation), then at certain time instant was at the border of the polygon (*touches* relation) and is within the region to the rest of the time period (*Within* relation). Similar definitions hold for *Leaves* (inverse of *Enters*), *Crosses* and *Bypasses* predicates.

The implementation of the proposed mobile object data model in an object-oriented LBS application and a mobile objects database is straightforward using defined UML class diagrams. The data model and the operators can be fully implemented on top "off the shelf" existing Object-Relational DBMS especially those supporting OGC Simple Features for SQL specification [6]. If the LBS information server is based on a relational DBMS, support for mobile object data management is implemented within the LBS application components. Whatever the implementation is, those operators are used to query the mobile object database, and to set triggers (or alerts) that are fired when interesting conditions are satisfied by the database during the objects motion. They can be incorporated into the traditional SQL query language that has been widely adopted by commercial database systems.

Having defined user-defined data types and operations related to mobile objects, the following feature types could be defined:

```
CREATE TYPE Ambulance AS OBJECT
(ID NUMBER, name  CHAR(64), driver  CHAR(64), type  CHAR(256), loca-
tions MobilePoint);

CREATE TYPE Taxicab AS OBJECT
(ID NUMBER, company  CHAR(64), driver  CHAR(64), type  CHAR(256), lo-
cations MobilePoint);

CREATE TYPE Street AS OBJECT
(ID NUMBER, name CHAR(64), centerLineOf LineString);
```

Based on them, the LBS client could define and specify the following queries:

1. Select ambulances that are within 2 km around of my current position:

```
select amb.id, amb.name, amb.driver
from Ambulance amb
where (amb.locations.GeometryAt(NOW)).distance(Point(xref,yref)) <=
2000
```

2. Select all taxicabs that are currently in "Vozdova" street:

```
select tcab.id, tcab.name, tcab.driver
from Taxicab tcab, Street s
where (tcab.locations.GeometryAt(NOW)).within(s.centerLineOf) and
s.name="Vozdova"
```

3. Return the position of a taxicab "Banker-17" if it enters the "Cara Dusana" street in last 5 minutes:

```
select t.locations.GeometryAt(NOW)
from Taxicab t, Street s
where t.name = "Banker-17" and s.name=" Cara Dusana" and
(t.locations.GeometryDuring(TimePeriod(NOW-
300,NOW))).Enters(s.centerLineOf))
```

4. Select all ambulances that were in "Niska Banja" municipality today between 5 and 6 AM.

```
select amb.id, amb.name, amb.driver
from Ambulance amb, Municipality m
where (amb.locations.GeometryDuring(TimePeriod(5, 6))).within(m.area)
and m.name = "Niska Banja"
```

All these queries, except the third one, are range queries, as the most prevalent type of queries in LBS. Therefore, the next section propose algorithms and data structures for processing of continuous range queries.

4 Processing Continuous Range Queries

The main problem and the challenge in the location-based services development is how to handle different types of queries over mobile objects. According to the mobility of clients and the data objects to be queried by the clients, such queries can be further classified into three types:

- Mobile objects querying static objects (tourist services, m-commerce, etc.)
- Stationary clients querying mobile objects (fleet management, traffic control, etc.)
- Mobile clients querying mobile objects (tourist services, digital battlefield, etc.)

The most prevalent types of location-dependent queries in LBS are range queries. In such queries the range can represent a user-selected area, a map window, a polygonal feature or an area containing points within specified distance from a point of interest. Since the motion of the mobile object is constrained by the road network, the range can also represent the part of the road segment. In contrast to regular queries that are evaluated once, a continuous query remains active over period of time and has to be continuously evaluated during this time to provide up to date answers. Such queries may also represent triggers that enable event notification to the registered users. At any time there will be several continuous queries running at the LBS server. Each of this queries needs to be re-evaluated as the objects move. A major challenge for this problem is repeatedly processing all queries within reasonable amount of time as the numbers of objects being tracked and continuous queries increases. We base our approach on the application scenario appropriate in LBS for tourist and business guiding, where mobile client's destination is known when he/she registers to the service, and also the underlying road network contains data specifying average speed and travel time on certain road segments.

The processing of the continuous range queries on mobile objects we base on the ARGONAUT, mobile object data management framework. We focus on the first and second types of queries, since the research on the third type can be based on solutions to the first two types. For the purpose of query processing, instead of 3D indexing scheme (3D R-tree) similar to the ones proposed in the literature [7], which requires frequent updating as the objects' move, we assume an available 2D indexing scheme in the underlying DBMS. The insertion of a mobile object's trajectory is done by enclosing every segment of the object's route in a Minimal Bounding Rectangle (MBR) and the same is done for the query range MBR. During the filtering stage the query engine retrieves the routes, which have a MBR that intersects with query range MBR. By exclusion of time in the indexing scheme we avoid the frequent index update, which occurs in the 3D indexing scheme. However, our filter stage as the result gives more objects that will cause false selections. In our work we focus on the refinement

stage of the processing and propose main-memory data structures for efficient query processing.

Continuous range query has to be continuously evaluated during specified time period, i.e. it has to be repeatedly and periodically processed. The shorten this period is, the more accurate is the query answer, but processing requires more time and computing power. In continuous query processing the filter stage is performed only once for initial query evaluation, but for every periodical evaluation the refinement stage must be performed. In order to shorten the refinement stage, which occurs repeatedly during the lifespan of the continuous query, we propose the pre-refinement step, which has to be performed just after the initial filter stage. Such pre-refinement step enables significant reduction of the refinement processing, by using expected trajectory of the mobile object and constructs the set of mobile objects that satisfies the query condition but possibly at different times during the period of the query time span.

The query issued by stationary client querying mobile objects we model by the *SMRangeQuery* class (fig 3).

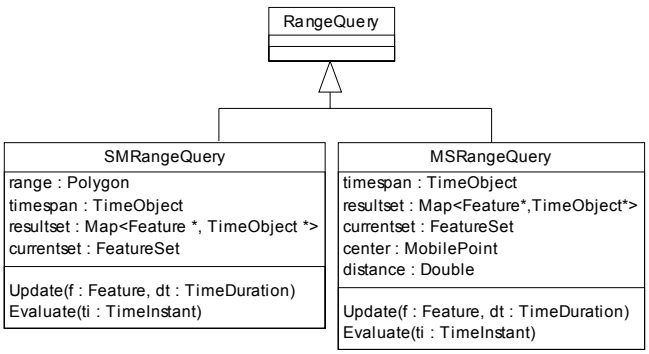

Fig. 3. Classes for range queries

The main attributes of that class are the *range* of the type of *Polygon* class, the *timespan* of the *TimePeriod* class representing the validity period of the continuous query, and the *resultset* of the *Map* class. The *Map* class represents the hash table that maps keys to values (e.g. *map* template class in STL, and *CMap* class in MFC). The key in the *resultset* attribute is a reference to the mobile object (*Feature* class with mobile geometry), and the value is the time object during which that mobile object satisfies the query condition. The *currentset* represents the set of features satisfying the query at the current (the time of evaluation) time instant. The algorithm for pre-refinement step assumes that the *filterset* contains the list of the mobile features selected by the filter stage, and is given by the following (Remark: Algorithms are given using C++ syntax and using MFC collection classes):

Algorithm 1 - The pre-refinement step for stationary clients querying mobile objects

```
POSITION pos = filterset.GetHeadPosition()
while (pos != NULL)
{
Feature *f = filterset.GetNext(pos);
```

```
MobilePoint *mp = (MobilePoint *) f->Geometry();
if(mp->Route()->interests(query.range))
{
            TimeObject *to = mp->When(mp->Within(query.range));
            query.resultest.SetAt(f, to);
}
}
```

For every mobile object in the filterset, this algorithm examines the predicate intersects between the route of the mobile object and the query range and, if the predicate is satisfied, inserts the new (key, value) pair to the query resulset, where the key represents the reference to the mobile feature, and the value is the time object (time period or multi time period) during which the feature satisfies the query condition. The refinement step of such continuous range query is preformed periodically on predefined time interval, and at each evaluation generates the current result set valid till the next evaluation. It is implemented as the *Evaluate* operation in the *SMRangeQuery* class.

Algorithm 2. The *Evaluate* operation and the refinement step at time instant *te*

```
POSITION pos = resultset.GetStartPosition()
while (pos != NULL)
{
resultset.GetNextAssoc(pos, f, to);
if(to->Contains(te))
            currentset.Add(f);
}
```

The refinement step iterates through all the elements in the result set, and adds to the current result set only those features whose time value contains time of the evaluation. Every mobile object has a reference to the set of continuous range queries in which it is involved (*queryset*). At every location update, the future trajectory of the mobile object must be updated by Δt - the time duration of its advance or delay related to the predicted trajectory (as noted in section 2). Also, the continuous query result sets in which that object is included must be updated accordingly. The following algorithm describes such an update:

Algorithm 3. Update of the query *resultset'*s time object for all the queries in the *queryset*

```
POSITION pos = queryset.GetHeadPosition();
while (pos != NULL)
{
            SMRangeQuery *rq = (SMRangeQuery *)queryset.GetNext(pos);
            rq->Update(f,  dt);
}
```

The *Update* operation of the *SMRangeQuery* class is defined as:

```
SMRangeQuery::Update(Feature *f, TimeDuration dt)
{
if(Lookup(f, tobject))
              SetAt(f, tobject->Add(dt));
}
```

Similar algorithms are implemented for the queries issued by mobile client querying stationary objects (points of interest) by using *MSRangeQuery* class (Fig. 3).

Since the mobile range in LBS is usually defined by the extent around the mobile object, it is represented by the mobile point (the attribute *center*) and the distance (the attribute *distance*) around it. The three other attributes, *timespan*, *resultset* and *currentset* have the same meaning as in the *SMRangeQuery* class. The pre-refinement step of such query is given by the following, assuming that the filter set contains all points of interest that are contained in the MBR of the query range:

Algorithm 4 - The pre-refinement step for mobile-static query

```
POSITION pos = filterset.GetHeadPosition()
while (pos != NULL)
{
Feature *f = filterset.GetNext(pos);
Point *p = (Point *) f->Geometry();
LineString *r = query.center->Route();
Polygon *area = r->buffer(query.distance);
if(area.contains(p))
{
TimeObject *to = rcenter->When(rcenter->Distance(p)-
>Lt(query.distance) );
            query.resultset.SetAt(f, to);
}
}
```

This algorithm calculates the buffer around the center point's route on the specified distance and examines the predicate contains between that polygonal buffer and the point from the filter set. If the predicate is satisfied it inserts the new (key, value) pair to the query result set, where the key represents the reference to the static feature, and the value is the time object (time period or multi time period) during which the feature satisfies the query condition. Algorithms 2 and 3 are also related to this type of query; only instead of the *SMRangeQuery* class, the *MSRangeQuery* class is used.

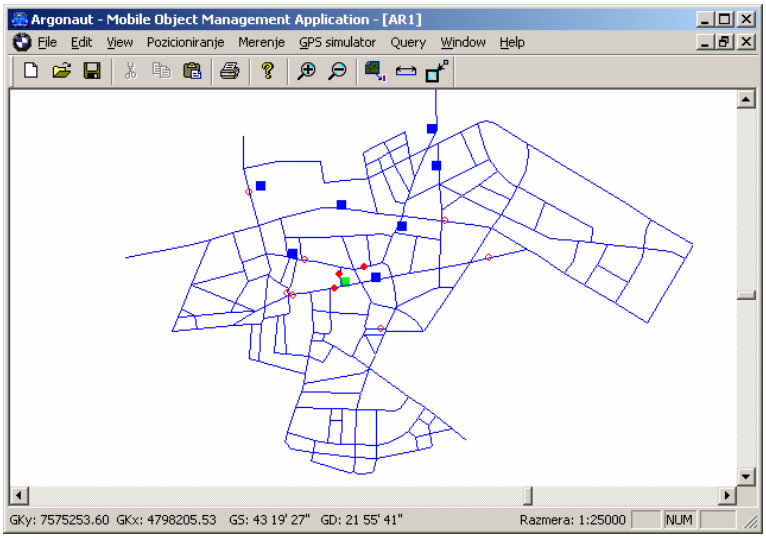

Fig. 4. Processing of continuous range query in the ARGONAUT prototype application

We've implemented proposed classes and data structures for continuous range query processing within the ARGONAUT framework. Using mobile objects' data generated by our Mobile Object Simulator [10], we are currently evaluating proposed continuous query processing algorithms, for a large number of mobile and stationary objects and compare their performance with general approach that rest on 3D indexing scheme and does not include the pre-refinement step. The processing of continuous range query, that from the set of all vehicles (red circles) selects only those (filled red circles) that are within the distance of 200 m around the specified point of interest (green rectangle) during specified time period, is shown in Figure 4.

5 Conclusions

In this paper we address the problem of processing continuous range queries on mobile objects. In contrast to regular queries that are evaluated once, a continuous query remains active over period of time and has to be continuously evaluated during this time to provide up to date answers. Such queries may also represent triggers that enable event notification to the users issuing them. At any time there will be several continuous queries running at the server. Each of these queries needs to be re-evaluated periodically as the objects move. A major challenge for this problem is repeatedly processing all queries within reasonable amount of time as the numbers of objects being tracked and continuous queries increases.

The processing of the continuous range queries on mobile objects we base on the mobile object data management framework, ARGONAUT, which represents an implementation of the object-oriented data model and the query language for mobile objects. We assume an available 2D indexing scheme in the underlying DBMS, and focus on the refinement stage of the processing proposing the pre-refinement step, algorithms and data structures which reduce computing time and complexity associated with periodical refinement procedure. The future research direction include improvements of the continuous query processing for the k-nearest neighbor queries, as well as practical evaluation of proposed query processing methodologies in the prototype LBS application for tourist and business guiding based on the ARGONAUT component framework.

References

1. Brinkhoff T.: A Framework for Generating Network-Based Moving Objects. *GeoInformatica* Journal, 6(2):153-180 (2002)
2. Erwig M., Schneider M.: Spatio-Temporal Predicates. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):881-901 (2002)
3. Gueting R.H., Boehlen M.H., Erwig M., Jensen C.S., Lorentzos N.A., Schneider M., Vazirgiannis M.: A Foundation for Representing and Querying Moving Objects. *ACM Transactions on Database Systems*, 25(1):1-42 (2000)
4. ISO/ TC 211 Geographic Information/Geomatics: ISO 19125 - Geographic information - Simple feature access. (2000)
5. ISO/ TC 211 Geographic Information/Geomatics: ISO 19108 - Temporal Schema. (2000)

6. Oracle 9i Enterprise Edition: Oracle Documentation Library. Oracle Corporation, http://technet.oracle.com, (2002)
7. Pfoser D., Theodoridis Y., Jensen C.S.: Novel Approaches in Query Processing for Moving Object Trajectories. *Proceedings 26th VLDB Conference*, (2000) 395–406
8. Stojanovic D., Djordjevic-Kajan S.: Location-based Web service for Tracking and Visual Route Analysis of Mobile Objects (in Serbian). *Proceeding Yu INFO Conference*, Kopaonik, (2002)
9. Stojanović D., Djordjevic-Kajan S.: Extending Database Technology to Support Location-Based Service Applications. *Proceedings ICEST Conference*, Sofia, Bulgaria (2003)
10. Stojanovic D., Djordjevic-Kajan S.: Generating Motion Data for Mobile Objects on Transportation Network (in Serbian). *Proceeding Yu INFO Conference*, Kopaonik (2003)
11. Trajcevski G., Wolfson O., Zhang F., Chamberlain S.: The Geometry of Uncertainty in Moving Objects Databases. *Proceeding EDBT Conference*, (2002) 233-250
12. Trajcevski G., Wolfson O., Xu B., Nelson P.: Real-Time Traffic Updates in Moving Objects Databases. *Proceedings DEXA Conference* (2002)
13. Wolfson O.: Moving Objects Information Management: the Database Challenge. *Proceeding 5th Workshop on Next Generation Information Technologies and Systems (NGITS),* Israel (2002)