Triggers and Continuous Queries in Moving Objects Databases

Goce Trajcevski
Dept. of Computer Science
University of Illinois at Chicago
Chicago, Il 60607
gtrajcev@cs.uic.edu

Peter Scheuermann *
Dept. of Electrical and Computer Engineering
Northwestern University
Evanston, Il 60208
peters@ece.nwu.edu

Abstract

This work addresses the problem of maintaining the consistency of the answers to continuous *queries which are posed by the users of the Moving Objects Databases (MOD). We propose a framework which enables detecting and processing the pending queries whose answers need to be re-evaluated upon modifications to the MOD. Relevant syntactic elements of the user's queries and their semantic implications are identified, and the basic components of a system that can be used for this task are specified. We show how* triggers *can be used to maintain the answers to the users' queries "up to date" with respect to the modifications to the MOD and we demonstrate that our system can be implemented on top of the existing ORDBMS.*

Keywords: Moving Objects Databases, Continuous Queries, Triggers.

1 Introduction and Motivation

A wide range of applications (traffic control, transportation industry, digital battlefields, environment monitoring, dynamic resource discovery, ...) need some form of a management of the *location* information [12]. The ability to store and process information about moving objects has spurred a lot of recent scientific research, where the subject is termed *Moving Objects Databases* (MOD) [23, 1, 14]. On the other hand, there is a bulk of work that has been done in the field of Active Database Systems. The seemingly straightforward event-condition-action paradigm, which adds a reactive behavior to the database systems, has been investigated from many aspects [5, 22]; and prototype systems have been implemented (e.g. STARBURST [21], Chimera [3]). Due to the recent trend for supporting *universal applications*, commercial Object-Relational Database Management Systems (ORDBMS) are now offering new (application-specific) complex data types; inheritance; user-defined routines which implement operators/methods over the user-defined types; extensions to SQL ([8]), predicates (e.g. *intersects, contains*) and functions for spatial calculations (e.g. *distance*).

The above observations point out a strong existing body of work which, so far, seems uncorrelated. In this paper we tackle an important problem in the MOD using active rules (triggers) and we demonstrate that the proposed framework can be implemented using the existing ORDBMS.

1.1 Problem Description and Our Contributions

Consider a MOD which stores the information about the trajectories of (a set of) moving objects. The MOST model in [15], identified three categories of queries: – *instantaneous*, for which the answer is evaluated immediately and transmitted to the user; – *continous*, which need to be evaluated at every "clock-tick" so that the consistency of their answer is ensured; – *persistent*, which not only need to be evaluated at each time instance, but may also require evaluation over an unbounded history.

In this work we focus on continuous queries like, for example *Q1: "Retrieve all the vehicles which will be no further than 0.3 miles from me, between 8:00PM and 8:10PM"*. If the query was posed at 6:30PM, it becomes continuous one, because many modifications to the MOD can occur between the time $Q1$ was posed and the relevant time-interval for its answer. For example, one of the cars that was part of the answer to $Q1$ changed its motion plan at 7:45PM. Not all the modifications to the MOD may be relevant to $Q1$, e.g. an accident at 7:30PM, on a road segment which affects *no* vehicle that is part of the answer of $Q1$.

*Research partially supported by NSF grant EIA-0000536

Our goal is to maintain the information about pending continuous queries to the MOD and avoid re-evaluation of their answers when it is not necessary, under the standard modifications: Updates, Deletions and Insertions. The main contributions of this work are:

- We identify the relevant syntactic elements of the users' continuous queries and their semantic impacts.
- We propose a framework which can be used to detect the set of queries whose answers are affected by the modifications to the MOD.
- We describe the specifications of the triggers which enable the MOD to properly react to the modifications (and avoid re-evaluation of continuous queries when not needed).

2 Preliminary Background

In this section we briefly introduce the model of the trajectory and its construction, and the issues related to the modifications of the MOD.

2.1 Trajectory Model and Updates

In order to capture the spatio-temporal nature of a given moving object, we need to, somehow, represent its motion. This information pertains to the object's whereabouts at a given time instance – *(location,time)*, and is represented using a *trajectory* [18]:

- *A* trajectory *of a moving object is a piece-wise linear function* $f : T \rightarrow (x, y)$, *represented as a sequence of points* (x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n) $(t_1 < t_2 < ... < t_n)$. *For a given a trajectory* Tr, *its projection on the X-Y plane is called the* route *of* Tr.

The object is at (x_i, y_i) at time t_i, and during each segment $[t_i, t_{i+1}]$, it moves along a straight line from (x_i, y_i) to (x_{i+1}, y_{i+1}), and at a constant speed. The *expected location* of the object at any time $t \in [t_i, t_{i+1}]$ $(1 \leq i < n)$ is obtained by a linear interpolation.

Relative to *now*, a trajectory can represent both the *past* and the *future* motion of objects. The future motion plan is constructed by using electronic maps[1] and the speed profiles information, which are input to the time-dependent shortest path algorithm [4] (see [18] for details). For the *past* motion of the object(s), one can use a set of 3D points (x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n), generated by the on-board GPS, which were transmitted by a moving object periodically. The points are "snapped" on the map, and then connected with the time-dependent shortest path.

This model of a trajectory can be represented as a User-Defined Type (UDT) in an ORDBMS. `trajectory` is a row type `LIST` of `point`, which is another row type having `X,Y` and `T` attributes:

```
MOT(oid,trajectory, ...other static attributes)
```

In the rest of this paper, MOT denotes the table which stores the moving object trajectoris and MOD denotes the moving objects database.

2.2 Modifications to the MOT

There are few sources which can cause modifications to the MOT, that we consider:

1. *insert* – At any time instance, a new trajectory may be inserted in the database, which is assigned a unique *oid*.

2. *delete* – An existing trajectory of a given moving object (given *oid*) may be deleted from the MOT.

3. *update* – There are two basic sources of updating a trajectory:

3.1. – A given moving object may decide to change its route.

3.2. – There may be some unforseen variations to the speed profiles used for constructing a future trajectory: accidents; road-works; bad weather; etc... In this case, the MOD server needs to utilize some sort of a real-time information to keep the (location-time) information accurate[2]. The details of *identifying* the trajectories affected by traffic incidents and their *updates* are presented in [17].

3 Classification of MOD Queries

In this section we analyze the important aspects of the requests that a user can pose to the MOD. We identify the *significant time*-aspects as syntactic elements in the queries, as well as the categories of queries.

We distinguish between two continuous *Query Requests (QR)* to the MOD: **1.** *Query Requesting Notification (QRN)* in which the user basically requests from the MOD to notify her/him when certain event occurs/ certain condition becomes true, like QRN_1 : *"Notify me when I am within 2 miles from hospital, between 1PM and 3PM".* **2.** *Query Requesting Answer (QRA)* in which an anwer-set needs to be transmitted to the user. For example, QRA_1 : *"Retrieve the motels that will be no further then 1 mile from my route, between 9PM and 10PM".*

3.1 Significant Times of the Requests

There are three *significant* time-instances pertaining to a given QR:

[1] Like, for example, the ones of Geographic Data Technology.

[2] For example, *(www.ai.uic.edu)* maintains the current traffic information for the expressways around Chicagoland.

• *Time Posed* – t_p, which is the time at which the QR is sent to the MOD.
• *Time to Answer* – t_a, which is the time at which the user wants the answer-set transmitted. For example, the user may pose a QRA_2 at 4PM: "Retrieve all the motels that will be no further then 1 mile from my route, between 9PM and 10PM, and *send me the answer* at 6PM" ($t_a = 6PM$).
• *Termination Time* – t_t, which is the time after which the QR is no longer valid. In the context of QRA_2 above, this time is $t_t = 10PM$.

3.2 Categories of Queries

Now we present the categories of queries which can be used in a QR. We do not address here the issues of their linguistic constructs or processing [14, 18].
• **Location Queries:** These queries pertain to the objects' whereabouts and time. We consider two variants:
1. *Where_at(t,oid)* – returns the *expected location* (i.e. the (x, y) coordinates) of the object *oid* at time t.
2. *When_at(x,y,oid)* – returns the *time* at which the object *oid* is expected to be at location (x, y).
• **Range Queries:** These spatiotemoral queries return the set of moving objects which are within a given (static) region, for a given time-interval. The basic syntax is $Inside(R, t_1, t_2)$.
• **Within_Distance Queries:** These queries have a *query_object* as one parameter and return a set of *answer_objects* (syntax is $Within\_Distance(obj, t_1, t_2)$) Based on the the nature (Dynamic or Static) of the *query_object* and the *answer_objects*, one can have for variants of this type of query: DD, DS, SD and SS (a spatial query).
• **k-Nearest_Neighbour(k-NN) Queries:** With the standard semantics – return the k nearest objects to a given object and, again with the 4 variants.

4 Basic Components

In this section we define the basic architecture which is needed to maintain the information about queries posed to the MOD and to properly update it upon modifications to the MOT.

4.1 Schemas

We extend the MOT schema, introduced in Section 2, with two more attributes – *Pending Posed Queries (ppq)* and *Part of Query Answer (pqa)*. Both of the new attributes serve as flags: *ppq* of the object oid_i is 0 if and only if there is *no* query posed by oid_i which is still "pending". Otherwise, the value of the *ppq* is the number of pending queries posed by the object oid_i; the *pqa* for a given oid_i is 0 iff there are no queries (posed by a mobile or web-based user) for which oid_i is part of their answer-set. $ppq = n$ denotes that the given oid_i is part of the answers for n different queries.

We have two more relations[3], `Issued` and `PartAnswer` with their respective schemas:
`Issued(User_id,Query_id,Terminate)` and
`PartAnswer(Query_id,Object_id)`.
The first relation keeps track of which user (recall that we can have web-based users) issued a particular query, and its t_t parameter. The second one maintains the information about objects which are part of the answer for a given query.

4.2 Scripts

We have several PL/SQL (or, equivalently, Embedded SQL) scripts:
1.) `TransmitAnswer(Q,U)` is used to transmit the answer of the query Q to the user U, who posed it. It extracts the t_a parameter of a given query `qid`. `SELECT`s all the `Object_id FROM` the `PartAnswer WHERE PartAnswer.Query_id = qid` and send them to the *uid*.
2.) `Receive(U,Q)`, upon receiving the query Q from the user U (*uid*) assigns a unique *qid* to Q; and inserts the tuple `(uid,qid,t)` into the table `Issued`. If *uid* is a mobile user, it also increments by one the value of its *ppq* attribute in the MOT. Then, it invokes the script `Eval(qid)` (see below) and creates an instance of the script `TransmitAnswer(qid,uid)`.
3.) `Eval(Q)`, basically evaluates a query *qid*. It properly updates the `PartAnswer` table with all the `(qid,oid)` tuples, where *oid* is an element of the answer-set for Q. If this was the first invocation of `Eval(Q)` (i.e. `PartAnswer(Query_id,Object_id)` did not have any tuples with *qid*), then the value for the *pqa* attribute of each *oid* in the MOT table is incremented by one. Otherwise: – if the tuple `(qid,oid_i)` was in the `PartAnswer`, but `oid_i` is no longer in the answer-set for Q, the tuple is deleted, and the *pqa* value for `oid_i` in the `MOT` is decremented by one; – if the tuple `(qid,oid_i)` was not in the `PartAnswer`, the tuple is inserted and the *pqa* value of the `oid_i` is incremented by one in the MOT.
4.) `Eval_All()` simply scans the `Issued` relation and, for every query *qid* whose `Terminate` attribute is not less than $t_{current}$, invokes the `Eval(qid)`.
5.) `Eval_All_Issued(UID)` scans the `Issued` relation

[3] We assume that the "static" data i.e. hospitals, motels, landmarks are properly represented and can be queried/accessed, say w.r.t. names and geo-coordinates.

and, for every tuple for which `Issued.User_id = UID` and `Issued.Terminate` > $t_{current}$, invokes `Eval(Issued.Query_id)`. If its execution caused modifications to the `PartAnswer` and there is an existing instance of the script `TransmitAnswer(QID,UID)` for which `QID = Issued.Query_id`, the new answer set is transmitted to the user `UID`.

6.) `ReEval_Answer(OID)` scans the relation `PartAnswer`. and for every tuple for which `PartAnswer.Object_id = OID`, it invokes the script `EVAL(PartAnswer.Query_id)`.

7.) `Remove(QID)` removes the tuple for which `Issued.Query_id = QID`; decrements the `MOT.ppq` counter for the respective `MOT.oid = Issued.User_id` (if the tuple is still in the `MOT`; removes every tuple from `PartAnswer` table for which its respective attribute `PartAnswer.Query_id = QID` and decrements the `MOT.pqa` attribute of the corresponding `MOT.oid = PartAnswer.Object_id`. Finally, it removes any existing instance of the `TransmitAnswer(QID,UID)`.

8.) `Purge(Issued)` periodically checks the `Issued` table. For every tuple for which the value of the `Terminate` attribute is less than $t_{current}$, it invokes the `Remove(Issued.Query_id)` script.

5 Active Rules

This section describes the syntax and semantics of the respective active rules which ensure that the changes to the queries' answers are properly considered.

5.1 Updates to the MOT

Recall that an update may be initiated by the mobile user itself or by the MOD server, when unexpected traffic conditions are detected from a real-time traffic site. To capture this, we have two triggers:

```
CREATE TRIGGER MOD_UPDATES_1
AFTER UPDATE OF Trajectory ON MOT
WHERE MOT.ppq > 0
Eval_All_Issued(MOT.oid) and:
    CREATE TRIGGER MOD_UPDATES_2
    AFTER UPDATE OF Trajectory ON MOT
    WHERE MOT.pqa > 0
    ReEval_Answer(MOT.oid)
```

The triggers are designed so that they capture both cases: the updated object has posed a query to the *MOD*; the updated object is part of an answer for a query posed by another object. We illustrate the behavior of the triggers with the following example, illustrated on Figure 1:

Example 1. *Assume that the moving object* oid4 *posed the following query at 1PM:* "retrieve all the objects which will be no further than 0.25 miles from me between 3:50PM and 4:00PM, and send me the answer at 3PM", *and it was assigned* Query_id = qid7. *When its answer was calculated, it had the objects* oid6 *and* oid9. *Now, suppose the Real-Time traffic site reported a congestion on certain road segments at 3:30PM. The* MOD *will identify the affected trajectories and update them accordingly (c.f. [17]). The thicker portions of the* oid5 *and* oid6 *on Figure 1 illustrate the slow-down update. This is an event which "awakes" both triggers. Since the moving object* oid6 *has not posed any queries itself, the condition part of the trigger* MOD_UPDATES_1 *fails. However, since* oid6 *is part of the answer to the* qid7 *(posed by the* oid4*), the condition for the* MOD_UPDATES_2 *is satisfied. Thus, its action is executed, which invokes the script* ReEval_Answer(oid6). *The sequence of execution is illustrated by the numbers in the circles above the arrowed lines. After executing* Eval(qid7) *it is detected that, due to the slow-down,* oid6 *is no longer part of the answer set. However, the object* oid5 *which, initially, was moving too fast to be "...no further than 0.25 miles from me (=* oid4*) between 3:50PM and 4:00PM..." (c.f. specifications of* qid7 *), now is slowed down enough so that it becomes part of the answer. Since the output generated by* Eval(qid7) *has changed, the new answer is transmitted to the user* oid4.

5.2 Deletions to the MOT

When a certain tuple is deleted from the `MOT` table (e.g. the moving object has a serious engine problem and it will not a traffic participant), again we need to consider its effects to the pending user's queries. Any query posed by the affected object itself is no longer of interest:

```
CREATE TRIGGER MOD_DELETIONS_1
AFTER DELETE ON MOT
          REFERENCING OLD AS OldTuple
WHERE (OldTuple.ppq > 0 )
REMOVE(X) WHERE (X IN
                 SELECT Query_id
                 FROM Issued
                 WHERE Query_id = OldTuple.ID)
```

In case the deleted object either has not posed any queries itself, or the ones that it posed are alredy removed from the `Issued` table, due to the execution of the `MOD_DELETIONS_1` trigger above (for which we always assign higher priority), we have:

```
CREATE TRIGGER MOD_DELETIONS_2
```

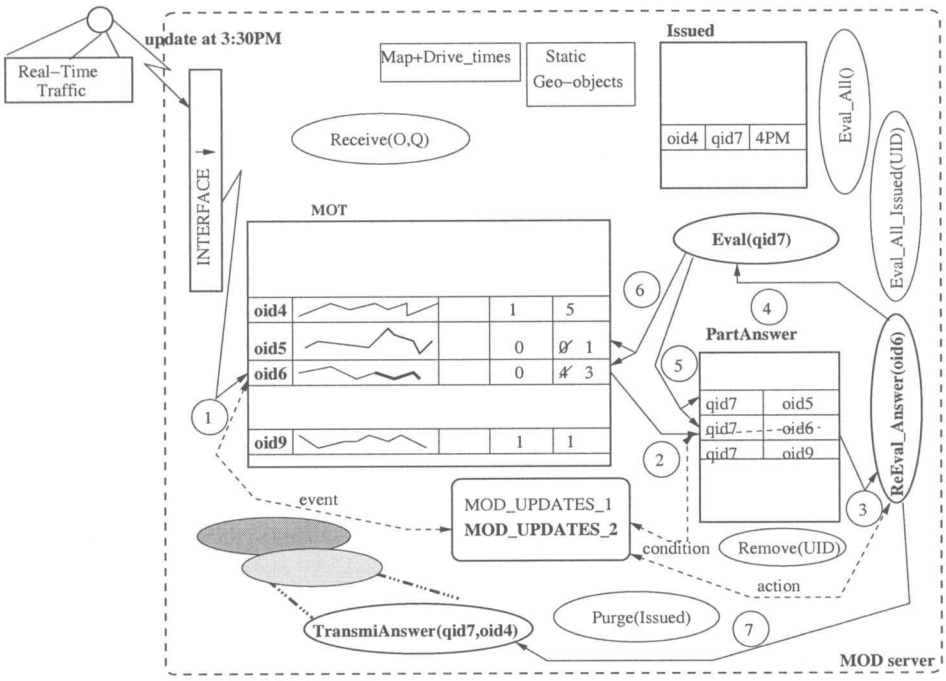

Figure 1. Triggers upon update to MOD

```
AFTER DELETE ON MOT
        REFERENCING OLD AS OldTuple
WHERE (OldTuple.pqa > 0 )
ReEval_Answer(OldTuple.ID)
```

In case the deleted object had both posed queries AND was part of some other query's answer, it is handled by the first trigger, `MOD_DELETIONS_1`, and the second trigger has no effect because all the tuples for which `PartAnswer.Object_id = OldTuple.id` are already deleted.

5.3 Insertions to the MOT

An insertion of a new moving object could possibly affect the answer set to every pending query in the `MOD`. Thus, we have the following:
`CREATE TRIGGER MOD_INSERTIONS`, for which:
- **Event:** `AFTER INSERT ON MOT`
- **Action:** `Eval_All()`

This situation is most straightforward to specify, but most complicated for the efficient processing of (its impact on) the pending users' queries (brute force approach is to simply re-evaluate every query in the `MOD`). The optimization problem is beyond the scope of this work and an ongoing research topic.

6 Related Work

The topic of active databases has been extensivelly studied for a long time [5, 22] (see the collection in [10] for an extensive list of references). Many aspects have been investigated: – *termination and confluence* [19]; – *coupling modes* between transactions which generated events vs. condition evaluation and actions execution [5]; – *expressiveness* and *behavior* issues [11, 2]. However, at the time when the research in the field of Active Databases was at its peak, the research on Moving Objects Databases was barely at its infancy. Due to the specific nature of the spatio-temporal domain, none of the works can be applied directly to the MOD settings.

Moving Objects Databases research has attracted a lot of attention in the recent years. Researchers have identified and investigated several aspects: – *Access Methods:* both in primal [14, 16] and in dual space [1, 7]; – *Uncertainty*: its communication cost vs. imprecision trade-off [23], and implication on the spatio-temporal range queries [18]; – *Linguistic issues and models:* modeling and querying moving objects by presenting a rich algebra of operators and data types [6] and the special case of road networks [20].

The MOST model [15] represented moving objects as a function of *(location, velocity_vector)* and introduced the notion of dynamic attributes and continuous queries, which are the topic of our work. As defined,

continuous queries required re-evaluation with every clock-tick and the algorithms use rather "esoteric" language (FTL-based). Our advantage is that we identified the important time-parameters and the catagories of queries and, based on their semantics, proposed a methodology which utilizes triggers to avoid evaluation of the continuous queries with every clock-tick.

7 Concluding Remarks and Future Work

We presented a framework for evaluating continous queries posed to the MOD. After presenting the categories of queries and their relevant "semantic dimensions": – time instances and dynamic vs. static nature of query_objects and query_answers, we proposed a framework which, upon modifications to the MOD will re-evaluate only the queries whose answer-set may be affected by those modifications. We identified the basic elements of the architecture (scripts and data tables) which can be implemented on top of the "off the shelf" ORDBMS.

Our ongoing research is targeted towards efficient execution of re-evaluation of the queries' answers, in a similar spirit to [13] and incorporating the context-awareness in the query processing [9].

References

[1] A. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *19th ACM PODS Conference*, 2000.

[2] C. Baral, J. Lobo, and G. Trajcevski. Formal characterization of active databases: Part ii. In *DOOD*, 1997.

[3] S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. Active rule management in chimera. In J. Widom and S. Ceri, editors, *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.

[4] S. E. Dreyfus. An appraisal of some shortest – path algorithms. *Operations Research*, 17(3), 1969.

[5] P. Fraternali and L. Tanca. A structured approach for the definition of the semantics of active databases. *Transactions on Database Systems*, 20(4), 1995.

[6] R. H. Güting, M. H. Böhlen, M. Erwig, C. Jensen, N. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and queirying moving objects. *ACM TODS*, 2000.

[7] D. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *18th ACM PODS Conference*, 1999.

[8] Oracle Corporation. *Oracle8i: Spatial Cartridge User's Guide and Reference, Release 8.0.4*, 2000. http://technet.oracle.com/docs/products/oracle8/doc-index.htm.

[9] A. Pashtan, R. Blatter, A. Heusser, and P. Scheuermann. Catis: A context-aware tourist information system. In *International Workshop on Mobile Computing (IMC)*, 2003.

[10] N. Paton, editor. *Active Rules in Database Systems*. Springer-Verlag, 1999.

[11] P. Picouet and V. Vianu. Semantics and expressiveness issues in active databases. In *Priniciples of Database Systems*, 1995. full version 1996.

[12] E. Pitoura and G. Samaras. Locating objects in mobile computing. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(4), 2001.

[13] S. Prabhakar, Y. Xia, D. Khalashnikov, W. Aref, and S. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE - TKDE*, 51(10), 2002.

[14] S. Saltenis and C. Jensen. Indexing of moving objects for location-based services. In *Intl. Conf. on Data Engineering, ICDE*, 2002.

[15] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *13th Int'l Conf. on Data Engineering (ICDE)*, 1997.

[16] J. Tayeb, O. Ulusoy, and O. Wolfson. A quadtree – based dynamic attribute indexing method. *The Computer Journal*, 41(3), 1998.

[17] G. Trajcevski, O. Wolfson, B. Xu, and P. Nelson. Real-time traffic updates in moving object databases. In *MDDS (in conjunction with DEXA)*, 2002.

[18] G. Trajcevski, O. Wolfson, F. Zhang, and S. Chamberlain. The geometry of uncertainty in moving objects databases. In *8th International Conference on Extending Database Technology (EDBT)*, March 2002.

[19] S. Urban, M. Tschudi, S. Dietrich, and A. Karadimce. Active rule termination analysis: An implementation and evaluation of refined triggering graph method. *JIIS*, 12(1), 1999.

[20] M. Vazirgiannis and O. Wolfson. A spatiotemporal model and language for moving objects on road networks. In *SSTD*, 2001.

[21] J. Widom. The starburst active database rule system. *IEEE Transactions on Data and Knowledge Engineering*, 8(4), 1996.

[22] J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.

[23] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7, 1999.