Reactive Maintenance of Continuous Queries *

Goce Trajcevski
goce@ece.northwestern.edu

Peter Scheuermann
peters@ece.northwestern.edu

ECE Department, Northwestern University, Evanston, IL 60208, USA

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). Assuming that the motion of the object is represented by a* trajectory*, we focus on the effect that the modifications to the trajectory data can have on the queries answer-set. In case a mobile user enters a road section in which an accident has occurred, which was not anticipated in the "expected" traffic behavior, not only his trajectory needs to updated, but the answer to the query that he posed may need to be recalculated and transmitted again.*
In this work we propose a framework which enables detecting and processing the pending queries whose answers need to be re-evaluated upon modifications to the MOD. We identify the relevant syntactic elements which can be extracted from the user's queries and we analyze their semantic implications. We also propose an architecture of a system that can be used for this task. We demonstrate 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 show that our framework can be implemented on top of the existing ORDBMS.*

I. Introduction and Motivation

A wide range of applications (traffic control, transportation industry, digital battlefields, ecology/ environment monitoring, mobile communication systems, dynamic resource discovery, ...) need some form of a management of the *locations* of the entities involved [21]. The ability to store and process information about moving objects has spurred a lot of recent scientific research in the field of *Moving Objects Data bases* (MOD) and generated a large body of results in several topics of interests: – modeling and querying the locations of moving objects [27, 36, 40]; – indexing schemas for efficient retrieval/update [1, 12, 25, 30]; – efficiency of location management and query processing subject to uncertainty [19, 40, 34].

On the other hand, there is a bulk of work that has been done in the field of Active Database Systems. Various aspects (e.g., expressiveness, termination and confluence, executional semantics [6, 10, 14, 28, 38]) of the seemingly straightforward event-condition-action paradigm, which adds a reactive behavior to the database systems, have been investigated and some prototype systems have been implemented (e.g. STARBURST [37], Chimera [5]).

On the commercial part, due to the recent trend for supporting *universal applications*, 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 ([3, 16]), predicates (e.g. *intersects, contains*) and functions for spatial calculations (e.g. *distance*).

The above observations point out important existing bodies of work which, so far, seem uncorrelated and one of the goals of this work is bring them together in order to address a significant and practically relevant problem. In particular, we tackle the problem of maintaining the correctness of the continuous queries in MOD settings and we show that this can be achieved using active rules (triggers). We also propose an architectural framework which can be implemented on top of the existing ORDBMS.

I.A. Problem Description and Our Contributions

Consider a MOD which stores the information about the *(location,time)* information for a set of moving objects. Due to the dynamic nature of the entities involved, the queries the one can pose to the MOD have been classified as (c.f. [27]): – *instantaneous* ones, for which the answer is evaluated immediately and transmitted to the user; – *continuous* queries, which

*This paper is an extended version of the paper *Triggers and Continuous Queries in Moving Objects Databases* that appeared in MDDS 2003. This research is partially supported by NSF grant IIS-0325144.

need to be evaluated at every "clock-tick" so that the consistency of their answer is ensured; – *persistent* queries, 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. For example, a user on-board a moving vehicle may be interested in: *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 (e.g. a company truck was re-routed) at 7:45PM. However, not all the modifications to the MOD may be relevant to $Q1$. If there is an accident at 7:30PM, on a road segment which affects *no* vehicle that is part of the answer of $Q1$, then the updates of the trajectories of the affected moving objects should not cause re-evaluation of $Q1$.

This is exactly the goal of our work – how to provide a reactive behavior which will enable the MOD to automatically maintain the correctness of the answers to pending continuous queries, and avoid re-evaluation of their answers when it is not necessary. We consider the standard modifications to the MOD (updates, deletions and insertions) and our main contributions can be summarized as follows:

- We identify the relevant syntactic elements in the users' continuous queries and analyze their (semantic) implications on the reactive maintenance of the correct answers.
- We describe the specifications of the triggers which enable the MOD to properly react to the modifications, for the purpose of (avoiding) re-evaluation of continuous queries.
- 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 set of functions/procedures (for which, in the sequel, we use the generic term *scripts*) needed for this functionality and we demonstrate how they can be incorporated on top of an ORDBMS.

The rest of this paper is structured as follows. Section 2 presents preliminary background and Section 3 outlines the categorization of the requests that a user can pose to the MOD and identifies the relevant syntactic elements. In Section 4 we describe the basic building blocks of our proposed architecture. Section 5 discusses the specifications of the triggers which, upon modifications to MOD, generate the activities needed for re-evaluation of the continuous queries. In Section 6 we briefly address the issue of possible optimizations of the reactive behavior, based on the information available in the requests' syntax. In Section 7 we position our work with respect to the existing literature and in Section 8 we conclude and outline directions for future work.

II. Preliminary Background

In this section we introduce the model of the trajectory and its construction and we address some issues related to the modifications of the MOD, which will be of interest for the subsequent sections.

II.A. 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* [34]:

• *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* XY *plane is called the* route *of* Tr*.*

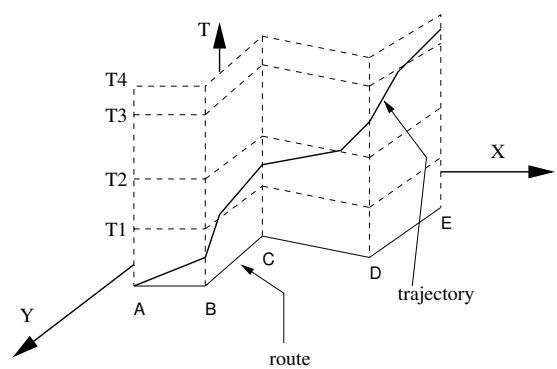

Figure 1: Trajectory and its construction based on a route and speed profiles

Thus, the object is at (x_i, y_i) at time t_i, and during each segment $[t_i, t_{i+1}]$, the object 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 between (x_i, y_i) and (x_{i+1}, y_{i+1}). An illustration of trajectory and its route is shown in Figure 1.

Relative to *now*, a trajectory can represent both the *past* and the *future* motion of objects. The future part of a trajectory corresponds to a *motion plan* of the

moving object and in order to explain how we construct it, we need to define an *electronic map*[1]:

• *A* map *is a graph, represented as a relation where each tuple corresponds to a block with the following attributes:*

– Polyline: *Each block is a polygonal line segment. Polyline gives the sequence of the endpoints:* (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)

– Length: *Length of the block.*

– Fid: *The block* id *number*.

– Drive_Time: *Typical drive time from one end of the block to the other.*

The GDT maps provide a single *Drive_time* attribute for a given block. However, this attribute is "static", in the sense that it does not consider variations of the traffic (e.g. the speed in a given block is 35 mph during "regular" hours, and 20 mph during "rush" hours). By monitoring the traffic, one can generate a *Speed_Profile* for a given block which is used in constructing the trajectory.

In order to construct the *future* part of the trajectory, the moving object transmits its *start_point* and *start_time*, and the *destination_point* (plus, possibly, a sequence of other "to-be-visited" points). Given the *start_time*, we use a *time-dependent* variant of the shortest path algorithm, as presented in [7](essentially, an A^* extension of Dijkstra's algorithm), where the cost of an edge in a graph depends on the start time to travel along that edge. Using this, we generate the shortest (in travel-time or distance) path between the *start_point* and the *destination_point*, and for each straight line segment, we compute the time of object's arrival at the end of the segment.

As far as the *past* motion of the object(s) is concerned, 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) which were transmitted by a moving object periodically, during its past motion (e.g. using an on-board GPS to detect location at given time points). To construct the trajectory, we first "snap" the points on the road network, and then connect the snapped points with the time-dependent shortest path.

Observe that, due to the Speed_Profile attribute used in the trajectory's construction, a single route_segment may yield more than one segment on the moving object's trajectory. This is because while being still within one particular route_segment (block), the object enters time-points at which the value of the Speed_profile for that route_segment has changed. The scenario is illustrated by the route segments $\overline{BC}, \overline{CD}$ and $\overline{DE}$ in Figure 1.

Let us point out that this model of a trajectory can be represented as a User-Defined Type (UDT) in an ORDBMS. We define the `trajectory` as a row type `LIST` of `point`, which is another row type having `X,Y` and `T` attributes. Thus, we have a schema for representing moving objects trajectories

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

In the rest of this paper we will use MOT to refer to the (instance of the) relation which stores the data about the moving object trajectories. The term MOD will denote the server which, besides managing the *(location,time)* information of the moving entities, also stores the information about static objects of interest (e.g., landmarks), map data, etc. and is used for communication with the moving objects. The components of the MOD which are relevant for maintenance of the continuous queries will be described in details in Sections 4 and 5.

II.B. 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. The request may come from a new moving object itself or a web-based user (i.e. a trucking company). In either case, the trajectory is generated as explained above and, after being assigned its unique oid, inserted in the MOT.

2. *delete* – An existing trajectory of a given moving object (given oid) may be deleted from the MOT. For instance, a given truck needs a service which is expected to take a substantial time, or a given vehicle was involved in an accident and will not be a traffic participant for a while.

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

3.1. – A given moving object may decide to change its route. For example, a particular truck needs to be re-routed to a certain warehouse because of a new pick-up request. In this case, the future part of the trajectory for the given *oid* is re-constructed and inserted in the MOT (after deleting the "old" future part).

3.2. – Although one may consider the speed profiles when constructing a future trajectory, there may still be some unforeseen variations, due to: accidents; road-works; bad weather; etc... In this case, the MOD server needs to utilize some sort of a real-time information to properly update the motion plans of the objects whose trajectory is affected by the unexpected traffic conditions. Otherwise, the (location-time) information stored in the MOT will become inaccu-

[1]Such maps are provided by, among the others, Geographic Data Technology *(www.geographic.com)*.

rate. This kind of information is available for many metropolitan cities. For example, The University of Illinois at Chicago *(www.ai.uic.edu)* maintains the current traffic information for the expressways around Chicagoland (I-55; I-290; I-90/94; I-294) which is updated every 2 minutes. The sources of information are traffic sensors (detectors) which are mostly located on highways (e.g. toll booths) and intersections of major streets. This kind of information can be used to detect an occurrence of abnormal traffic conditions on given sections of a road network. In order to utilize this information, the MOD needs to: 1.) *Identify* the trajectories which are affected by the abnormal traffic; and 2.) *Update* the *(location-time)* information about their (future) motion plans. These issues were addressed in [33], where the model of a *traffic incident* was proposed, which is utilized to *update* the trajectories affected by abnormal traffic conditions. In order to *identify all* the trajectories which are affected by the abnormal traffic, the model was proposed which captures the spread of the effects of a traffic incident on the neghbouring road-segments (called the *traffic spill-over effect*). Due to lack of space, we will not elaborate on the technical details of [33] here. For the purposes of this work, we will assume that there is a methodology which can be used to detect the trajectories affected by some traffic incident and, based on the type of the incident, correctly updates their future portions.

III. Classification of MOD Queries

In this section we present the categories of queries which can be used in a user's request in our framework. and we identify the important syntactic elements that can be extracted from a given request.

We assume that the users *on-board* moving vehicle have some minimal processing power, so that they can formulate their queries/requests and receive the answer(s), e.g. they have a pocket-PC or a Personal Digital Assistant (PDA). The *web-based* users of the MOD are assumed to have some form of an interface for the same purposes.

We distinguish between two basic categories of continuous *Query Requests (QR)* to the MOD, as the simplest and the most sophisticated:

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[2]. An example of the is 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, a user on-board moving object may ask QRA_1 : *"Retrieve all the motels that will be no further then 1 mile from my route, between 9PM and 10PM"*.

Observe that there may be other requests where the user needs both notification and some answer transmitted. In the context of QRN_1, this would correspond to QR'_1 : *"Notify me when I am within 2 miles from the hospital and tell me the name of it"*. In the rest of this work, without any loss of generality, we will focus on the QRA types.

III.A. Categories of Queries

Now we present the categories of queries which can be used in a particular QR. We do not address the issues of their linguistic constructs or processing because various aspects of interest have already been investigated (c.f. [1, 25, 34, 36]).

• **Location Queries:** These queries pertain to the objects' whereabouts and time. We consider two types of location queries:

1. *Where_at(t,moid)* – returns the *expected location* (i.e. the (x, y) coordinates) of the object $moid$ at time t.

2. *When_at(x,y,moid)* – 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 region, for a given time-interval. The basic syntax is $Inside(R, t_1, t_2)$.

• **Within_Distance Queries:** The basic syntax is $Within\_Distance(obj, d, t_1, t_2)$ and these queries return a set of objects which are no further than d units from the object obj, within the time-interval $[t_1, t_2]$.

• **k-Nearest_Neighbor (k-NN) Queries:** The semantics of these queries is standard – for a given query_object, a set of k answer_objects is returned, constituting the k objects which are closest to the query object, within a given time interval. Their processing has already been investigated for spatio-temporal context [1, 23, 29] and the basic syntax is $NN(obj, k, t_1, t_2)$.

[2]Let us point that in the active database literature, two extreme alternatives are to have both the event and the condition embedded in the event and use a rich event-processing language [15]; or have both of them embedded in the condition and use some temporal logic-based processing [28].

III.B. Significant Times of the Requests

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

• *Time Posed* – t_p, which is the time at which the QR is sent to the MOD[3].

• *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". In this case, $t_p = 4PM$ and $t_a = 6PM$. In case the request is of a notification type (QRN), by default, we assume that t_a is the time instance at which the event of interest has occurred (equivalently, the condition becomes satisfied). Observe that, we may have more than one t_a value – in the context of QRN_1 the object may be within 2 miles from the hospital H_1 at 1:30PM and within 1.5 miles from the hospital H_2 at 1:55PM.

• *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 $t_r = 10PM$. In theory, if the user does not provide a termination time as a part of the request, the query may have an "infinite duration" and, in turn, require infinite re-evaluation. However, in practice, every trajectory has some time-instance t_{end}, at which the object has reached the end of its trip. By default, we assume that $t_t = t_{end}$.

With respect to the relevant time-interval of the query (e.g. 9PM to 10PM for QRA_1 above), the property of interest may be satisfied *sometimes* (resp. *always*) or within some sub-interval(s) of the given time-interval of the query. This corresponds to the $\exists$ (resp. $\forall$) quantifiers in the temporal domain or some percentage-based value of the validity of the property of interest. These aspects impact the processing of the respective queries, but not the main design aspects of our work and, without loss of generality, we assume throughout the rest of the work that the queries under consideration are under *sometimes* semantics of the temporal quantifier.

IV. System Architecture

As we mentioned in Section 1, according to the definition in [27], continuous queries require that their answer-set is re-evaluated with every clock-tick. Clearly, this is very impractical and imposes a lot of computational overhead on the MOD server. In this section we define the main components which are needed to maintain the information about queries posed to the MOD and to properly update it upon modifications to the MOT.

IV.A. Schemas

We extend the MOT schema, introduced in Section 2, with two more attributes – *Pending Posed Queries (ppq)* and *Part of Query Answer (pqa)*. Now the schema is:

`MOT(oid, trajectory, ..., ppq, pqa)`

Both of the new attributes serve as flags and their semantics is as follows:

• ppq of the object oid_i is 0 if and only if there is *no* query posed by the object oid_i which is still "pending" (i.e. may need a re-computation of its answer-set upon modifications to the MOT). Otherwise, the value of the ppq corresponds to the number of pending queries posed by the object oid_i.

• pqa for a given oid_i is 0 iff there are no queries (posed by any user – mobile or web-based) 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[4], `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 allow for a web-based users of MOD) issued which query, and what is the time after which the query is no longer valid, which is the t_t parameter of the query. The second one maintains the information about which object is a part of the answer for a given query.

IV.B. Scripts

We have several PL/SQL scripts:

1.) The first one, `TransmitAnswer(Q,U)` is used to transmit the answer of the query Q to the user U, who posed it. This is a trigger-like script (not a pure database trigger) which does the following: – for a given query qid, it extracts its t_a parameter. Recall that (c.f. Section 3), every query request has its t_a, which is the time at which the user wants the answer sent to him/her by the MOD. At t_a, the `TransmitAnswer(qid,uid)` instance of the script will `SELECT` all

[3] We do not distinguish between a valid time and transaction time.

[4] We do not discuss here any of the relations pertaining to the "static" data, i.e. hospitals, motels, landmarks. We assume that they are properly represented and can be queried/accessed w.r.t, as least, their names and geo-locations.

the `Object_id FROM` the `PartAnswer WHERE PartAnswer.Query_id = qid` and send the projection to the uid.

2.) The script `Receive(U,Q)`, upon receiving the query Q from the user U (uid): – assigns a unique qid to Q; – inserts the tuple `(uid,qid,t)` into the table `Issued`, where $t = t_t$ is the termination time of the qid. If the user uid is a mobile one, it also increments by one the value of its ppq attribute in the MOT. Subsequently, it invokes the script `Eval(qid)` (see below) and creates an instance of the script `TransmitAnswer(qid,uid)`.

3.) The `Eval(Q)`, basically evaluates a query, for a given qid. After that, 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 the 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)` script scans the relation `Issued` and, for every tuple for which `Issued.User_id = UID` and `Issued.Terminate >` $t_{current}$, it invokes `Eval(Issued.Query_id)`. If the execution of `Eval(Issued.Query_id)` caused modifications to the `PartAnswer` table 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)`, on the other hand, scans the relation `PartAnswer`. For every tuple for which `PartAnswer.Object_id = OID`, it invokes the script `EVAL(PartAnswer.Query_id)`. Similarly to the previous script, if the execution of `Eval(PartAnswer.Query_id)` caused modifications to the `PartAnswer` table and there is an existing instance of the script `TransmitAnswer(QID,UID)` for which `QID = PartAnswer.Query_id`, the new answer set is transmitted to the user `UID`.

7.) `Remove(QID)` is a script which: – removes the tuple for which `Issued.Query_id = QID`; – decrements the `MOT.ppq` counter for the respective `MOT.oid = Issued.User_id`), in case that 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`. The last action if the `Remove(QID)` script is to remove any existing instance of the `TransmitAnswer(QID,U)`.

8.) Finally, the script `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.

The concepts that we introduced are illustrated in Figure 2, which we use in the following:

Example 1. *Observe an instance of the MOD at 1PM, at which time the mobile user* $oid4$ *issues the query:* 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. *The query is assigned a* `Query_id` $qid7$. *The* significant times *for* $qid7$ *are* $t_p = 1PM$, $t_a = 3PM$ *and* $t_t = 4PM$. *The sequence of steps taken by the invocation/execution of the corresponding scripts and the appropriate modifications are indicated in the circles on the arrowed lines (the scripts that are activated are drawn with the thicker oval lines). Observe that the object* $oid4$ *had already posed another query* $qid2$ *which terminates at 2PM. Hence, its ppq attribute is incremented. Since the objects* $oid6$ *and* $oid9$ *become the answer to* $qid7$*, their respective pqa attributes are modified accordingly. As the user requested, the script* TransmitAnswer(qid7,oid4) *will be executed at 3PM. In the context of the terminology proposed in [29], the answer will be of the form:* ([oi6,(3:50, 3:55)], [oid9,(3:55, 4:00)]).

V. Active Rules

As one may have observed, Figure 2 contains an element labeled `TRIGGERS` which was not elaborated upon in the Example 1 above. In this section, we focus on the basic elements which provide the proper reactive behavior which guarantee that answers to the (pending) continuous queries to the MOD are kept up-to-date. We consider the standard modifications to the MOT, which are *UPDATE*, *DELETE* and *INSERT* and, as we mentioned, each of these modification may cause changes to the answer-set(s) of the users queries. We describe the syntax (and semantics)

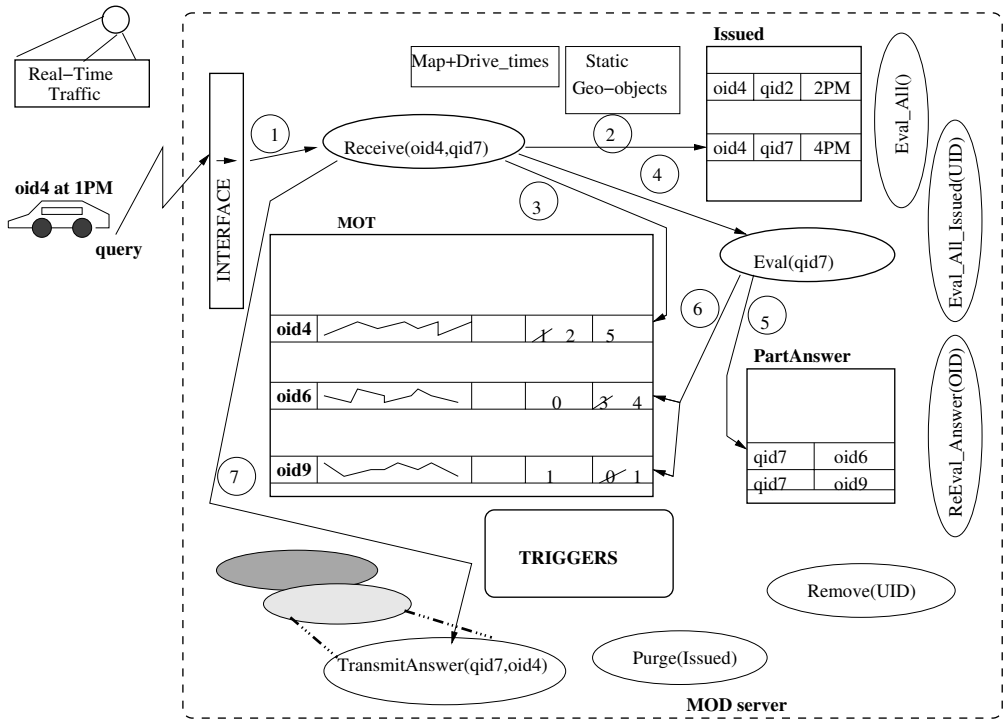

Figure 2: Query request posed by a mobile user to the MOD

of the respective active rules/triggers[5] which will ensure that the changes to the queries' answers are properly reflected. Since our goal is to utilize the capabilities of existing ORDBMS for their implementation, we follow the standard *Event* – *Condition* – *Action* (ECA) paradigm.

V.A. Updates to the MOT

Recall that an update may be initiated by: 1. The mobile user itself – either it decided to change its motion plan or its on-board computer detected that it deviates from its MOD-modeled trajectory (c.f. [34]); 2. The MOD server, when unexpected traffic conditions are detected from a real-time traffic site. To capture this, as a part of the MOD specifications we have two triggers. The first trigger is:

```
CREATE TRIGGER MOD_UPDATES_1
AFTER UPDATE OF Trajectory ON MOT
WHERE MOT.ppq > 0
Eval_All_Issued(MOT.oid)
```

and the second one is:

```
CREATE TRIGGER MOD_UPDATES_2
AFTER UPDATE OF Trajectory ON MOT
WHERE MOT.pqa > 0
ReEval_Answer(MOT.oid)
```

[5]We do not confine to any particular DBMS here, although our implementation uses *Oracle 9i*. Instead, our specifications are written, as generically as possible, in compliance with the SQL3.

The triggers are designed so that they capture both cases: the updated object has posed a query to the MOD and 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 2. *Figure 3 gives a (partial) instance of the MOD used in the Example 1, at 3:30PM. Observe, firstly, that the query* qid2 *is no longer in the* Issued *table because, as indicated on Figure 2, its termination time was 2PM, so it was purged from the MOD. Assume that the Real-Time Traffic site reported a congestion on a certain road-segment. The* MOD *will identify the affected trajectories and update them accordingly (c.f. [33]), as illustrated by the thicker portion of the* oid6 *on Figure 3. Due to the update to the* MOT*, we have 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*

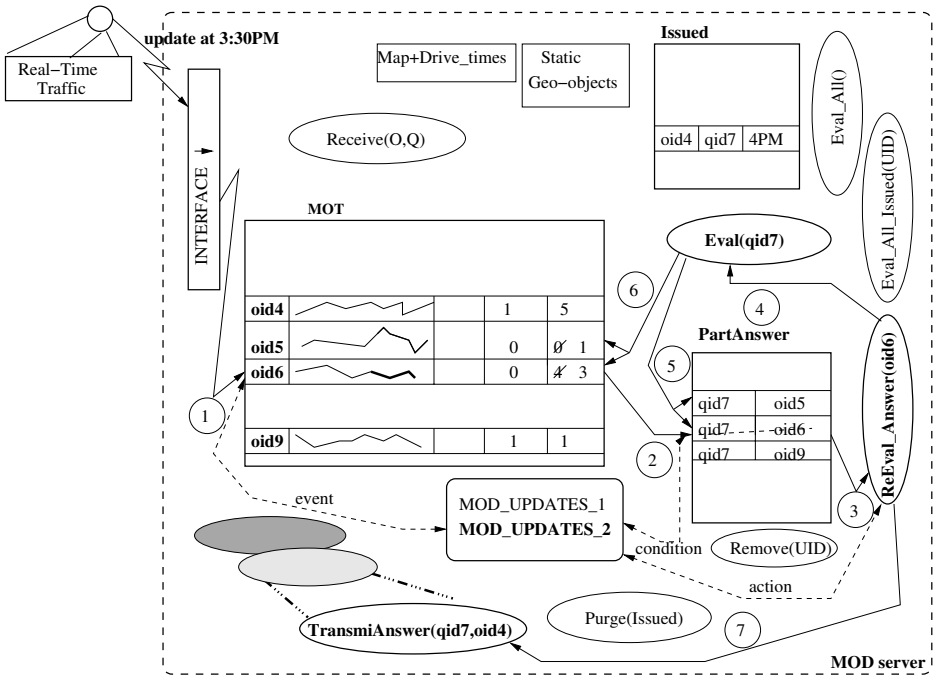

Figure 3: Triggers upon update to MOD

0.25 miles from me (= oid4*) between 3:50PM and 4:00PM..." (c.f. specifications of* qid7 *in the Example 1), 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* ([oid9, (3:52,3:57)], [oid5, (3:58,4:00)]) *is transmitted to the user* oid4.

V.B. Deletions to the MOT

When a certain tuple is deleted from the `MOT` table e.g., a particular moving object has a serious engine problem and will not be part of a traffic for a significantly long time, again we need to consider its effects to the pending user's queries in the `MOD`. Firstly, any query posed by the affected object itself, is no longer of interest. For that, we have:

```
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)
```

The second case occurs when the deleted object either has not posed any queries itself, or the ones that it posed are already removed from the `Issued` table, due to the execution of the `MOD_DELETIONS_1` trigger above (for which we always assign higher priority). However, we still need to consider the case when it was part of the answer to the query posed by another object. This is handled by:

```
CREATE TRIGGER MOD_DELETIONS_2
AFTER DELETE ON MOT
       REFERENCING OLD AS OldTuple
WHERE (OldTuple.pqa > 0 )
ReEval_Answer(OldTuple.ID)
```

Observe that, in case the deleted object had both posed queries AND was part of some other object's query, 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.

V.C. Insertions to the MOT

When inserting a new moving object to the `MOT`, the situation is most straightforward for the specifications of the trigger, but most complicated for the efficient processing of (its impact on) the pending users' queries.

Basically, the insertion of a new moving object could possibly affect the answer set to every pending query in the `MOD`. Thus, the brute force approach is to simply re-evaluate every query posed to the `MOD`. Clearly, this is very inefficient, but the optimization problem is beyond the scope of this work (and an on-

going research topic). We would like to point out that an attempt towards optimization, which could serve as a basis for "better - than - brute - force" approach is given in [22], however there are some limitations to the proposed methodology, which we address in Section 6. Thus, we have the following brute-force to ensure the correctness of the answers to the pending continuous queries:

```
CREATE TRIGGER MOD_INSERTIONS:
AFTER INSERT ON MOT
Eval_All()
```

VI. Possibilities of Syntax-Based Optimization of the Reactive Behavior

While the optimization of the reactive maintenance of the continuous queries was not a goal of this work, it is a topic of our ongoing research. In this section we would like to demonstrate how a careful analysis of the syntactic information available in the query requests can be used in conjunction with the options which are readily available in the commercial OR-DBMS which, in turn, may enable some form optimization of the reactive behavior.

Recall the categories of query requests that we considered in Section 3. A careful observation will reveal that a particular QR has:

• A **query-object** (QR_{qo}), and

• An **answer-object(s)** (QR_{ao}).

Similarly to the observations in [29], based on the combinations of the nature (Dynamic or Static) of the *query_object* and the *answer_object(s)*, we have the following options:

1. *DD* – Both the query_object and the answer_objects are moving. An example of this query would be: *" Retrieve all the moving objects which are within d = 5 miles from the moving object* mo_id*, between* t_1 *and* t_2".

2. *DS* – The query_object is moving, but the answer_objects are static. An example is QRA_1 (c.f. Section 3): *"Retrieve all the motels that will be no further then 1 mile from my route, between 9PM and 10PM"*.

3. *SD* – Here, the query_object is static, but the answer_objects are moving. An example of this query is: *" Retrieve all the objects which are within d = 5 miles from the monument* mid*, between* t_1 *and* t_2". Observe that this can be viewed as a special case of a range query.

4. The last combination *SS* is a pure spatial query.

To illustrate the possible impact that the combination of the *Type* of a particular QR can have in conjunction with its *Dynamics*, let us consider the following:

QRA_3 *"Retrieve all the moving objects which will be inside the region R between 3:30 and 3:45PM"*.

Assume that the query was posed at *t_p = 1:30PM* and that its answer was calculated to be $A(QRA_3) = \{o_2, o_3, o_8, o_{10}\}$. Also, assume that at 3:00PM a traffic abnormality occurred which affected all of the trajectories of the moving objects which are part of the answer $A(QRA_3)$. As described in Section 5, upon updates to the MOT, a separate instance of the trigger `MOD_UPDATES_2` will be enabled and (eventually) fired for each of the objects o_2, o_3, o_8 and o_{10}. Clearly, this will bring the answer-set $A(QRA_3)$ up-to-date, however, a lot more efficient behavior can be achieved. Namely, instead of having one "generic" type of trigger for all updates, one could automatically generate an instance of a particular trigger type upon receiving a QR, which will be tailored towards syntax-based efficient reactive behavior. In the case of QRA_3 above, a trigger similar to `MOD_UPDATES_2` can be created upon receiving the QRA_3 which should be specified to execute in a *set-oriented* granularity. This will ensure that the respective trigger will fire after all the updates of o_2, o_3, o_8 and o_{10} have completed and is very likely to yield more efficient behavior than the same trigger executing in *row-oriented* granularity, where a separate instance of the respective trigger will be fired after the update of each individual trajectory in the answer $A(QRA_3)$.

Let us point out that QRA_3 is a *range* query and its dynamics is of a type *SD*. On the other hand, note that the request QRA_1 is of a *within_distance* type and its dynamics is *DS*. In such a case, generating a corresponding trigger which executes in a *tuple-oriented* manner when MOT is updated, could yield an execution which is as efficient as the one when the trigger is specified to execute in *set-oriented* manner.

VII. Related Work

The topic of active databases has been extensively studied for a long time [4, 6, 10, 38] (see the collection in [18] for an extensive list of references). Many aspects have been investigated: – *termination and confluence* [35]; – *coupling modes* between transactions which generated events vs. condition evaluation and actions execution [10]; – *event processing and con-*

sumption [15]; – *expressiveness* issues [20] and semantics of the active rules behavior, for which several formalisms have been used, like for example, action theories [2], temporal logic [28] (to name a few). 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. Most of the queries, as well as the action parts of the triggers, are dependent on User-Defined Functions (UDF) for their processing.

Moving Objects Databases research has attracted a lot of attention in the recent years. Researchers have identified and investigated several aspects: 1.) *Access Methods:* much work has been done on selecting an appropriate index for certain types of queries both in primal [24, 26, 25, 30] and in dual space [1, 12, 13]. Specifications of relevant "dimensions" for a MOD indexing are presented in [31, 32]; 2.) *Uncertainty*: [39, 40] introduce a cost based approach to determine the size of the uncertainty for optimizing the communication cost and query imprecision. Another model of uncertainty and its implication on the spatio-temporal queries is given in [34], whereas [19] gives a quantitative characteristics of the objects whereabouts for a given boundary of the speed along a road segment; 3.) *Linguistic issues and models:* Series of works [8, 9, 11] address the issues of modeling and querying moving objects by presenting a rich algebra of operators and a comprehensive framework of data types. [36] work presents new operators for special cases of spatio-temporal range queries.

The MOST model [27] introduced the notion of dynamic attributes (similar to the approach in [26]) and introduced the notion of continuous queries, which are the topic of our work. As defined in [27], continuous queries require re-evaluation with every clock-tick. In this work, we identified the important time-parameters of the categories of queries and, based on their semantics, proposed a schema which utilizes triggers to avoid evaluation of the continuous queries with every clock-tick, and demonstrated that the framework can be readily implemented on top of the existing ORDBMS.

A recent approach towards efficient evaluation of queries in a dynamically changing spatio-temporal environment is given in [22]. The authors present an interesting view where the queries are indexed, instead of the usual indexing of the moving objects, which enables incremental evaluation of continuous queries. They restrict their focus to range queries where the regions are bounded by rectangles. We see these results as something that could be utilized in the future when we focus on the efficiency of the triggers' execution.

The impact that the dynamics of the objects involved has on the efficient processing of the spatio-temporal queries has been addressed in [29]. However, the authors do not consider any reactive ability to handle the changes to the (stored) representation of the dynamic attributes.

VIII. Concluding Remarks and Future Work

We presented a framework for evaluating continuous queries posed to the MOD. After presenting the categories of queries and their relevant "semantic impact": – time instances and dynamic vs. static nature of query_objects and query_answers, we proposed a system which, upon modifications to the MOD will re-evaluate the 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.

The work that we presented here is part of a larger research effort that we are currently undertaking in the area of context-aware MOD. Although we never explicitly mentioned it (and, as we only hinted in Section 6), the values that can be extracted from the syntax of the queries are nothing but values of a particular *context dimension* pertaining to a given query request. What is the interplay of the various context dimensions and how can that be utilized towards the overall efficiency aspect of the reactive behavior of the MOD? For example, the *Motion_Plan* can be viewed as a context dimension which can have a value of a type *trajectory* or of a type *sequence of GPS-reported points*. Each type can have a different impact on the processing and maintaining the continuous queries posed by the respective user, when taken in conjunction with the values of the *Query_Type* and *Dynamics* context dimensions. Moreover, one cannot disregard the impact of the values of some *Environmental* context dimension, e.g., the *Traffic_Conditions* may be *normal* or *congested* which, as we demonstrated in this work, affects the trajectories (*Motion_Plan*) and the answers to the pending continuous queries. One particular topic that we are focusing on is incorporating the sensor-generated data into the overall context-awareness settings. For example, a high temperature reported by a set of sensors may indicate a fire in a given region which, in turn, impacts the traffic condi-

tions and generates a scenario similar to the ones that we considered in this work.

The steps that we have taken towards the foundation of treating the continuous queries posed to the MOD, open some other interesting challenges for the future work. Part of our ongoing research is targeted towards system-level optimization of the reactive behavior, in the sense of efficient execution of re-evaluation of the queries' answers, in a similar spirit to [22]. Another long term goal is to investigate if/how incorporating the user's preferences, as another context dimension, can be utilized for optimizing the evaluation of the pending continuous queries, along the lines of the results presented in [17] and how to incorporate the uncertainty of the moving objects whereabouts into the framework (c.f. [34]).

References

[1] A. K. Agarwal, L. Arge, and J. Erickson. Indexing Moving Points. In *Proceedings of the ACM PODS International Conference*, 2000.

[2] C. Baral, J. Lobo, and G. Trajcevski. Formal Characterization of Active Databases: Part II. In *Proceedings of the International Conference on Deductive and Object-Oriented Databases (DOOD)*, 1997.

[3] M. Carey, D. Chamberlin, S. Narayanan, B. Vance, D. Doole, S. Rileau, R. Swegarman, and N. Mattos. O-O What Have They Done to DB2. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1999.

[4] S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. Automatic Generation of Production Rules for Integrity Maintenance. *ACM Transactions on Database Systems*, 19(3):367–422, 1994.

[5] 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.

[6] U. Dayal, E. Hansen, and J. Widom. Active Database Systems. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability and Beyond*. Addison–Wesley, 1994.

[7] S. E. Dreyfus. An Appraisal of Some Shortest-Path Algorithms. *Operations Research*, 17(3), 1969.

[8] M. Erwig, M. Schneider, and R. H. Güting. Temporal and Spatio-Temporal Datasets and Their Expressive Power. Technical Report 225-12/1997, Informatik berichte, 1997.

[9] L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. In *Proceedings of the ACM SIGMOD International Conference*, 2000.

[10] P. Fraternali and L. Tanca. A Structured Approach for the Definition of the Semantics of Active Databases. *ACM Transactions on Database Systems*, 20(4), 1995.

[11] 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 Transactions on Database Systems*, 25(1), 2000.

[12] D. Kollios, D. Gunopulos, and V. J. Tsotras. On Indexing Mobile Objects. In *Proceedings of the ACM PODS International Conference*, 1999.

[13] G. Kollios, D. Gunopulos, and V. J. Tsotras. Nearest Neighbour Queries in a Mobile Environment. In *Spatio-Temporal Database Management*, 1999.

[14] B. Ludaescher. *Integration of Active and Deductive Database Rules*. PhD thesis, Universitat Freiburg, 1998.

[15] I. Motakis and C. Zaniolo. Formal Semantics for Composite Temporal Events in Active Database Rules. *JOSI*, pages 1–37, 1997.

[16] 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.

[17] A. Pashtan, R. Blatter, A. Heusser, and P. Scheuermann. Personal Service Areas for Location-Based Wireless Web Applications. In *IEEE Internet Computing*, (to appear in Fall) 2004.

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

[19] D. Pfoser and C. Jensen. Capturing the Uncertainty of Moving Objects Representation. In *International Symposium on Advances in Spatial Databases (SSD)*, 1999.

[20] P. Picouet and V. Vianu. Semantics and Expressiveness Issues in Active Databases. In *Proceedings of the ACM PODS International Conference*, 1995.

[21] E. Pitoura and G. Samaras. Locating Objects in Mobile Computing. *IEEE Transactions on Knowledge and Data Engineering*, 13(4), 2001.

[22] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *Transactions on Knowledge and Data Engineering*, 51(10), 2002.

[23] G. Karciauskas R. Benetis, C. Jensen and S. Saltenis. Nearest Neighbor and Reverse Nearest Neighbor in Moving Objects. In *Proceedings of the IDEAS International Conference*, 2002.

[24] S. Saltenis and C. Jensen. R-tree Based Indexing of General Spatio-Temporal Data. Technical Report TR-45, TimeCenter, 1999.

[25] S. Saltenis and C. Jensen. Indexing of Moving Objects for Location-Based Services. In *Proceedings of the IEEE International Conference on Data Engineering*, 2002.

[26] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proceedings of the ACM SIGMOD International Conference*, 2000.

[27] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proceedings of the IEEE International Conference on Data Engineering*, 1997.

[28] P. Sistla and O. Wolfson. Temporal Conditions and Integrity Constraint Checking in Active Database Systems. In *Proceedings of the ACM SIGMOD International Conference*, 1995.

[29] Y. Tao and D. Papadias. Spatial Queries in Dynamic Environments. *ACM Transactions on Database Systems*, 28(2), 2003.

[30] J. Tayeb, O. Ulusoy, and O. Wolfson. A Quadtree-based Dynamic Attribute Indexing Method. *The Computer Journal*, 41(3), 1998.

[31] Y. Theodoridis, T. Sellis, A. N. Papadopoulos, and Y. Manolopoulos. Specifications for Efficient Indexing in Spatiotemporal Databases. In *Proceedings of the International Conference on Statistical and Scientific Database Management*, 1999.

[32] Y. Theodoridis, J. R. O. Silva, and M. A. Nascimento. On the Generation of Spatiotemporal Datasets. In *Proceedings of the International Symposium on Large Spatial Databases*, 1999.

[33] G. Trajcevski, O. Wolfson, B. Xu, and P. Nelson. Real-Time Traffic Updates in Moving Objects Databases. In *Proceedings of the MDDS workshop (in conjunction with the DEXA Conference)*, 2002.

[34] G. Trajcevski, O. Wolfson, F. Zhang, and S. Chamberlain. The Geometry of Uncertainty in Moving Objects Databases. In *Proceedings of the International Conference on Extending Database Technology*, 2002.

[35] S. Urban, M. Tschudi, S. Dietrich, and A. Karadimce. Active Rule Termination Analysis: An Implementation and Dvaluation of Refined Triggering Graph Method. *JIIS*, 12(1), 1999.

[36] M. Vazirgiannis and O. Wolfson. A Spatiotemporal Model and Language for Moving Objects on Road Networks. In *Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD)*, 2001.

[37] J. Widom. The STARBURST Active Database Rule System. *IEEE Transactions on Data and Knowledge Engineering*, 8(4), 1996.

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

[39] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and Imprecision in Modeling the Position of Moving Objects. In *Proceedings of the IEEE International Conference on Data Engineering*, 1998.

[40] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases*, 7, 1999.