Dynamic Topological Predicates and Notifications in Moving Objects Databases

Goce Trajcevski,Peter Scheuermann*
Department of ECE
Northwestern University
Evanston, Il 60208
goce,peters@ece.northwestern.edu

Hervé Brönnimann
Department of CIS
Polytechnic University
Brooklyn, NY
hbr@poly.edu

Agnès Voisard
Institut für Informatik
Freie Universität
Berlin, Germany
voisard@inf.fu-berlin.de

ABSTRACT

This work addresses the problem of efficient reactive management of topological predicates in MOD (Moving Objects Databases) settings. Detecting the satisfiability of such predicates in mobile and dynamic environments requires management of continuous and persistent conditions. We introduce two dynamical topological predicates: *moving_along* and *moving_towards* and we present efficient algorithmic solutions for their processing. Based on this, we subsequently take a deeper insight in the behavioral aspects of a MOD which manages them and we argue that the traditional ECA (Event Condition Action) paradigm, while it may ensure correct behavior, is not well-suited for enabling the users to *declaratively* specify some of the parameters that may affect the *efficiency* aspect of the reactive behavior. Towards this end, we introduce the $(ECA)^2$ (Evolving and Context-Aware Event-Condition-Action) paradigm as a tool for specification of the triggers used by a MOD that handles requests which span over a time-interval in dynamic environments.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*query processing*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*data types and structures, dynamic storage management*

General Terms

Algorithms, Languages

Keywords

Topological Predicates, Moving Objects Databases, Event Notification Systems

*Research partially supported by NSF grant IIS-0325144

MDM 2005 05 Ayia Napa Cyprus

1. INTRODUCTION AND MOTIVATION

Mobility is one of the most important aspects of applications like mobile tour guides, dynamic service discovery, on-time information delivery, digital battlefields [4, 8, 16, 21]. An essential enabling technology for these applications is *location management* [23, 25], which is, the management of the transient location information of the (mobile) objects involved. Another relevant aspect of these application is the concept of *context awareness*. First introduced in [24], and subsequently refined in [1], *context* is basically any information pertaining to a given situation, which can be used to characterize an entity. For example, the GUIDE project [8], which guides city visitors equipped with a hand-held device, distinguishes between *personal:* (preferences, location, attractions visited, etc); and *environmental:* (time of day, attraction's opening time, etc.) contexts. In a typical scenario of a person driving in a given region, the user has *static demands*, based on his profile (e.g. a type of restaurant, budget, etc.) and he needs to be notified when they are in certain proximity of his *evolving* position. However, that person may also have *dynamic information demands* which are functions both of his location and the dynamic environmental conditions (e.g., traffic congestion due to road accidents or bad weather), for which he needs to be notified too (c.f., IN:SIGHT system [19]). One can typically distinguish between two broad categories of approaches for managing requests for notification [25]. In a *pull mode*, an end user explicitly asks for information stored in the data sources, e.g., *nearest gas station* to his *current* location. In a *push mode*, the system *automatically* sends the relevant information, for which the user may have subscribe beforehand – according to the profile, context, and so on – without the user asking explicitly for it. In this case information filtering and delivery is handled by alerting systems or Event Notification Systems (ENS) [16]. However, existing systems [8, 16, 19, 21] have only limited capabilities of matching the dynamics of the location with a static set of preferences.

Location-based services require effective integration of information originating from a number of data sources, e.g., databases about mobile users, environmental conditions, etc. Clearly, one of the important components is the *Moving Objects Database* (MOD), which stores the (location,time) information of the mobile entities and has the ability to process various *requests* (query and/or notification) pertaining to the whereabouts-in-time of the entities involved. Database researchers have addressed many issues related to modeling

and querying moving objects (see the collection [18] and the references therein). However, the dynamics of the environment requires a *reactive* behavior of the underlying MOD with respect to the pending users requests, which was addressed only recently [20, 27, 29].

Unlike the traditional database applications where the queries are *instantaneous*, many queries of interest to MOD are *continuous* and/or *persistent* (they span over a time-interval and/or require re-evaluation over the history (c.f. [26]) and, due to the dynamics of the entities involved, their answers change over time.

The main goal of this work is to address some of the aspects of efficient management of the reactive behavior targeted towards notification requests in MOD settings, considering the impact of the *evolution* of various context dimensions. In order to focus the discussion, we analyze two predicates, *moving along* and *moving towards*, which are of interest not only for the typical ENS application scenarios [16, 19, 21] but also for applications like emergency response, homeland security and digital battlefield [4].

Our main contributions can be summarized as follows:
• We introduce two new predicates and we present efficient algorithms for their processing of based on adaptation of computational geometry techniques. We also address the efficiency of their managing from the perspective of the overall behavior of the MOD, and we introduce the concept of dynamic triggers.
• We demonstrate that the traditional approaches for handling the *moving along* and *moving towards* available in the existing commercial Object-Relational Database Management Systems (ORDBMS) are not suitable for ENS which depend on the *dynamics* of the objects *and* their relationship with the environment.
• We introduce a new paradigm for expressing reactive behavior in MOD. The paradigm is geared specifically towards heterogeneous environments, where the dynamic information may be coming from sensors, GPS devices, MOD, traditional databases changing with time, etc. As such, our model explicitly allows for self-modifying triggers which take into account the evolving relationships between values of different context variables. We call our paradigm Evolving and Context-Aware Event-Condition-Action – $(ECA)^2$

The rest of this paper is organized as follows[1]. In Section 2 we give a preliminary background and we identify the relevant aspects of our work in the context of our ongoing implementation of the *CAT++* system [28]. Section 3 introduces the semantics of the two dynamic topological predicates and addresses their efficient processing from two perspectives – *algorithmic* and *behavioral*, in the sense of the declarative specification of the reactive behavior of the MOD. These motivate the $(ECA)^2$ paradigm, which is subsequently introduced in Section 4. Section 5 positions our work with respect to the related works in the literature and Section 6 concludes the paper and presents our directions of future work.

[1]Due to lack of space, we omit the technical details, which can be found in [28], along with the full list of references. The report is available at *http://www.ece.northwestern.edu/ peters/contextMOD/*

2. PRELIMINARIES

Typical assumption of MOD settings is that each moving object is equipped with some minimal processing power, e.g., an on-board GPS to detect its location and the ability to transmit it, along with transmitting requests and receiving answers to/from the MOD. The representation of the object's motion in a MOD is based on some *model* of the transient *(location,time)* information and different application domains have adopted different models. Three models which are, in a sense, "extrema of the spectra" can be described as follows:
• The object is assumed to (periodically) send updates of its location using, for example, its on-board GPS. The characteristic of this model is that the future of the object's motion is unknown and, as a consequence, spatio-temporal queries pertaining to the future will have to be re-evaluated upon new updates [20]. The past motion of the object is represented as a sequence of 3D (2D geography + time) points of the form (x_i, y_i, t_i) with the intended meaning that the object was at the location with the coordinates (x_i, y_i) at the time-instant t_i. Typically, between two points the object is assumed to move in accordance with some evaluable function of time, e.g., along a straight line and with a constant speed in which case its motion is represented as a polyline in the 3D space.
• Instead of only sending its location, the object also transmits the information about its *velocity*, whenever it changes. This is represented as a *dynamic attribute* (c.f. [26]) in the MOD. In this model, a "near future" is assumed known and various queries can be posed pertaining to some future time (interval) of interest. However, upon update of the dynamic attributes representing the changes of the object's motion plan, the answers of the pending queries may need to be re-evaluated. Similarly to the previous case, the past motion is modeled as a 3D polyline, represented as a sequence of 3D points.
• Given the object's initial location and a set of the points that one intends to visit, by using an electronic map augmented with some information about the variations of the traffic patterns, one could construct the entire *future trajectory* of the object's motion plan (c.f. [30]. The MOD model is again a polyline except now one can pose queries pertaining to the *future* of the

In this work, the object is assumed to (periodically) send updates of its location. The future whereabouts of the object are unknown and its past motion is represented as a sequence of 3D (2D geography + time) points of the form (x_i, y_i, t_i). Between two points the object is assumed to move along a straight line and with a constant speed. We focus on Requests for Notification (**RN**) in a push-mode and, as usual, an *event* denotes an occurrence of "something of interest", e.g., a predicate becoming satisfied or a detection of an occurrence of external stimuli. We assume a distinct set of *primitive* events, which are pre-defined by the system [6]. An event's occurrence is assumed instantaneous, in a distinct time-value, and is registered with the system until deleted. We also assume availability of an *event algebra* which enables specifications of *composite events* using the primitive ones (e.g., $E = e_1; e_2 - sequence -$ meaning that E is detected whenever e_2 occurs, following an occurrence of e_1), with an underlying mechanism for their detection (e.g., Event Graphs [6]).

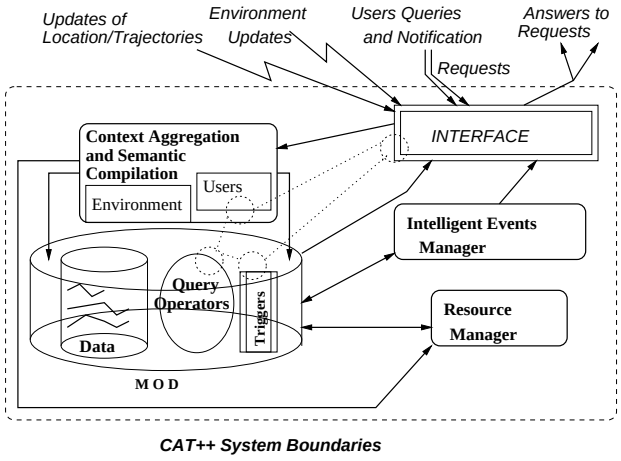

Figure 1: Topological Predicates and Declarative Specification of Reactive Behavior

Figure 1 illustrates the main components of our *CAT++* (Optimal Management of Correct Answers to Continuous and Persistent Requests Using Triggers) system, which we are currently implementing based on our CAT system [27, 29]. What is relevant for this paper, is that we propose the algorithms which implement the *operators* for processing of the dynamic topological predicates. We also address the issue of executing the *triggers* which correctly react to the updates of the moving objects' locations from the perspective of detecting the satisfiability of the predicates. However, we do so in a manner which is *aware* of the inter-relations among the values of different context dimensions and we allow the users to *declaratively* specify these triggers.

3. DYNAMICS OF THE TOPOLOGICAL PREDICATES AND REACTIVE BEHAVIOR

Now we introduce the semantics of the two dynamic topological predicates and for each of them we identify the issues relevant for their processing from two perspectives: *operational* (or, algorithmic) efficiency and *behavioral*, in the sense of efficiency of the reactive behavior of the underlying MOD.

3.1 The *Moving_Along* Predicate

In spatial settings, the *alongness* property has been investigated both from topological (the *9-intersection* model in [17]) and spatial database [13] perspective. When it comes to "alongness" in mobile environments, in reality one cannot expect that a mobile user, say, driving a car, can move *exactly* along a river. Thus, we introduce a distance threshold d with its intuitive meaning that for as long as the object is within distance d from a given 2D polyline P, we will assume that it is moving "along" it. We are also interested if the predicate is satisfied within a portion Δt of a time-interval $[t_1, t_2]$. As a particular example, consider the following request which is important in scenarios like homeland security and digital battlefield:

RN2: *"Notify me when the object obj_1 is moving along the polyline P and within distance d less than 90% of the time between 5:00 and 5:30"*, or, equivalently:

RN2': *"Notify me when the object obj_1 is moving along the polyline P and further than distance d for more than 10% of the time between 5:00 and 5:30"*

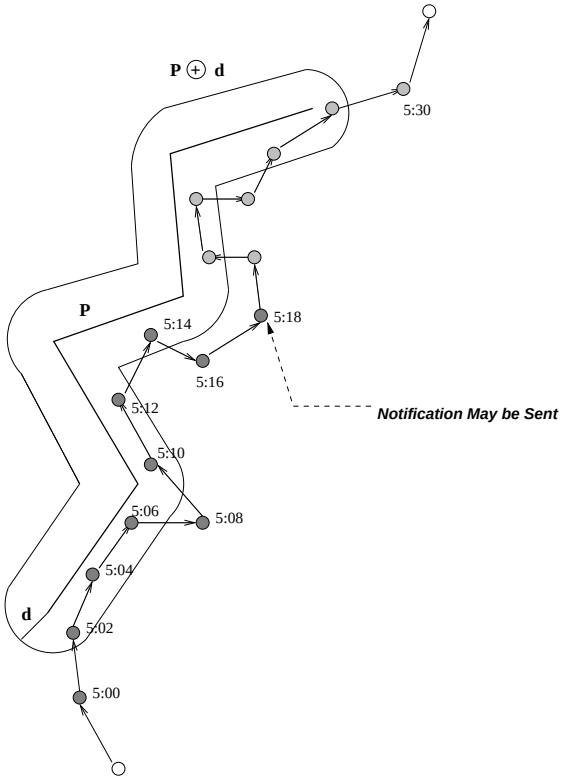

Figure 2: Notification for *Moving_Along*

Figure 2 shows an example scenario, where each circle indicates a *(location,time)* update sent to the MOD server. Assume, for this example, that they are sent every two minutes. Blank circles indicate the *(location,time)* pairs which are of no interest for processing **RN2** because the value of their *time* component is outside the time-interval of interest for **RN2** ([5:00,5:30]).

In order to determine the spatial region of interest for **RN2**, one can construct the *Minkowski Sum* $P \oplus d$ of the polyline P and a disk of radius d. Formally, given two sets in $\mathcal{R}^2$, say P_1 and P_2, their *Minkowski Sum*, denoted by $P_1 \oplus P_2$, is defined as $P_1 \oplus P_2 = \{p_1 + p_2 \mid p_1 \in P_1, p_2 \in P_2\}$, where the summation is of vector p_1 with vector p_2 [2]. In our case, this is the 2D region that is "swept" when the disk with radius d is moves along P. The evaluation of the *moving along* predicate amounts to calculating the time that obj_1 spent inside $R = P \oplus d$.

When it comes to processing **RN2**, we have the following observations:

1. Once the time-interval of interest expires (at 5:30), using a variant of the *red-blue* intersection problem over the completed past motion (c.f. [30]), one can detect the intersection points of the route (the 2D projection of the trajectory) of oid_1 with $P \oplus d$, and use linear interpolation to calculate the total time for which the object obj_1 was (not) within

distance d from P. If that time is less than 27 minutes (the object was ≥ 3 minutes outside $P \oplus d$), the notification can be send to the user. One may also use (a composition of) the readily available spatio-temporal operators (c.f. [12] and the references therein), whose processing is based on the *plane sweep* technique.

In practice, however, these kinds of approaches are inadequate, because they need the entire history of the objects motion throughout the time-interval of interest. They may be suitable for processing a query like: *"Retrieve the time that the object obj_1 is (not) moving along $P \oplus d$ between 5:00 and 5:30"*. However, this is not acceptable for mission-critical applications (e.g., detecting an enemy's activity in a battlefield). As indicated in Figure 2, the system should be able to notify the user as early as 5:18 that **RN2** is satisfied – by then, the object has already spent more than three minutes (10% of the time-interval) *outside* $R = P \oplus d$. Any further update (lighter-shaded circles in Figure 2) need not be considered.

2. One may want to utilize the capability of (re)active behavior available in most existing commercial systems and set up a trigger **TRN2**, for which the basic elements (in pseudo-syntax) are:

```
EVENT:      ON location_update(oid_1,x,y,t)
CONDITION:  IF time_outside  P ⊕ d ≥ 3
ACTION:     SEND NOTIFICATION
```

This will achieve the desired behavior – the user will be notified at 5:18, however, it also has drawbacks. At every update, the system evaluates the `time_outside` for the entire past trajectory between 5:00 and the time of the current update. An intelligent system should know that there is no need to repeatedly evaluate the condition of **TRN2** on the entire past trajectory.

3. One may be tempted to impose a spatial index [11] (e.g., a variant of the R-tree [14]) on the line-segments of P. Upon arrival of a new *(location, time)* update, the last segment of the object's route is embedded in a 2D rectangle which extends its endpoints by some distance d' and the index is used to determine the segments of P which intersect the bounding rectangle. Subsequently, the proximity of each of these segments of P to the object's route segment is evaluated and used to calculate the total time that the object was (not) within distance d from P. However, despite seeming the most "spatially-orthodox" approach, the main drawback of this idea is that unnecessarily many segments of P may be tested against the latest route-segment of the moving object. This, in turn, implies that the approach is not *output-sensitive*, which is our goal.

3.1.1 Algorithmic Processing of the Moving_Along Predicate

Given the segment between two consecutive location-update points, say (x_i, y_i, t_i) and $(x_{i+1}, y_{i+1}, t_{i+1})$, and the region $R = P \oplus d$, we would like to avoid the brute-force approach of testing $\overline{(x_i, y_i), (x_{i+1}, y_{i+1})}$ for intersection against every (line or circular arc) segment of the boundary of R. The algorithm for calculating the total time that the object moving along the segment $\overline{(x_i, y_i), (x_{i+1}, y_{i+1})}$ and with a constant speed between t_i and t_{i+}, spent inside R, can be outlined as follows:

Moving_Along(R**,** (x_i, y_i, t_i)**,** $(x_{i+1}, y_{i+1}, t_{i+1})$ **)**

1. *Decompose R*
2. *Shoot a ray from (x_i, y_i) towards (x_{i+1}, y_{i+1}) and obtain the intersections with R*
3. *Use linear interpolation to obtain the total time inside R*

Decomposition of R can be done in many ways (e.g., vertical, or a triangulation [9]), however, an arbitrary decomposition may cause the segment $\overline{(x_i, y_i), (x_{i+1}, y_{i+1})}$ to intersect too many edges of the decomposition, without ever intersecting the boundary of R. Thus, we propose to use a *Steiner* decomposition [15], based on a *geodesic triangulation* [7], which guarantees that, if n is the number of segments of the boundary of R, than *any* ray will intersect *at most* $O(\log n)$ edges before "hitting" the boundary. The Geodesic Decomposition Tree (GDT) structure [15] can be easily adapted to handle the circular arc segments on the boundary of R. In case R is non-simple, i.e., it is a multi-connected region with k simple components, the complexity becomes $O(\sqrt{k}\log n)$. Observe that there are two costs which are amortized while monitoring the *moving along* predicate:

1. The decomposition itself is used for consecutive updates (and may be re-used for other objects);
2. The point-location is executed only once, for the first *(location,time)* update. Subsequently, the terminus of the previous trajectory segment becomes the origin for the next one (for which the terminus is the latest update point).

Observe that if the predicate is satisfied after *m+1* updates (let T denote that portion of the object's trajectory), total complexity becomes $O(m + |T \cap R| \log n)$, which is output sensitive.

3.1.2 Behavioral Aspects of the MOD

At present, the existing prototype implementations of MOD and spatio-temporal databases (c.f. [3]) lack the mechanisms which would enable an elegant and declarative specification of the desired kind of reactive behavior. One may use the available extensibility features of the commercial ORDBMS (e.g., Oracle) , and use them as a MOD [30]. However, a straightforward application of their available triggers cannot achieve the reactive behavior that one would desire from a notification service. Besides the limited set of primitive events (update, insert, delete on tables), the very *model* of a trajectory, typically represented as a User-Defined Type (UDT) in an ORDBMS, has its own semantic subtleties. Namely, despite the lexical similarity, the classical meaning of the *update* (e.g., a salary of a particular employee) in relational settings, has different impact from *location update* of a particular moving object in MOD. The *net-effect* of a sequence of location updates increases the number of points (in the representation) of the trajectory of a given moving object. Meanwhile, a sequence of salary updates of a given employee has the net-effect of one single (lump) increase. To better illustrate the peculiarities of **RN2**, consider the request **RN1**, common for ENS:

RN1: *"Notify me when I am within 3 miles from a motel, between 7:00PM and 9:00PM"*.

Setting up a trigger **TRN1**, of the form:

```
EVENT:      ON location_update(oid,loc,t)
CONDITION:  IF within_distance(loc,Motel)
ACTION:     SEND NOTIFICATION
```

would yield a correct behavior, without the processing overhead exhibited by the similar trigger **TRN2** for **RN2**. However, in the case of **RN2**, the condition *evolves* along with the modifications to the MOD.

In order properly monitor the event of interest in an efficient manner, we need a trigger that follows the *dynamics* of the MOD's evolution. Let *TimeInside (R,* (x_1, y_1, t_1), (x_2, y_2, t_2),t^{in}*)* denote a predicate which is true when t^{in} is the total time between t_1 and t_2, during which the line segment $\overline{(x_1, y_1)(x_2, y_2)}$ was inside[2] R. Also, let *previous_last(OID)* and *last(OID)* denote the functions which return the (x, y, t) values of the next_to_last and the last point, respectively, in the representation of the *Motion_Plan* of the object OID. We have the following dynamic version of the trigger which monitors **RN2**:

TRN2dyn:
1. ON *location_update* (oid_1, x, y, t)
2. IF $TimeInside(R, last(oid_1), previous\_last(oid_1), t^{in})$
   $\wedge \quad t^{in} \geq (3 - t^{in}_{total})$
3. THEN
4. $\quad Send\_Notification$
5. ELSE $\quad t^{in}_{total} = t^{in}_{total} +$
   $TimeInside(R, last(oid_1), previous\_last(oid_1))$

The variable t^{in}_{total} is the accumulator which denotes the total time that the object oid_1 had spent moving along $R(= P \oplus d)$ from the begin-time value specified in the request **RN2**.

Clearly, in an actual implementation one would not evaluate the predicate $TimeInside(R, last(oid_1), previous\_last(oid_1))$ twice and a separate structure would be set to keep track of the t^{in}_{total} accumulator. However, that is a different issue from what we are trying to tackle in this work – our goal is to enable the users to *declaratively* specify the aspects of evolution of the MOD that are relevant for the efficient processing of the *moving_along* predicate.

3.2 The *Moving_Towards* Predicate

Now we consider a predicate which is concerned with detecting if a particular mobile object is *continuously moving towards* a given static entity like a point-object, region or a (poly)line. We would like to point out that a variant of the problem – the necessary and sufficient conditions on the object's trajectory for the purpose of missile guidance *towards* a point-target – has already been addressed by the control community [5]. To illustrate the aspects of the reactive behavior that we are investigating in this paper, we will use the following:

RN3: *"Notify me when the object obj_2 is moving towards the landmark* LM *continuously for 5 minutes between 5:00 and 5:30".*

[2]In reality, t^{in} is the value returned by the algorithm *Moving_Along (R,* (x_1, y_1, t_1), (x_2, y_2, t_2)*)*.

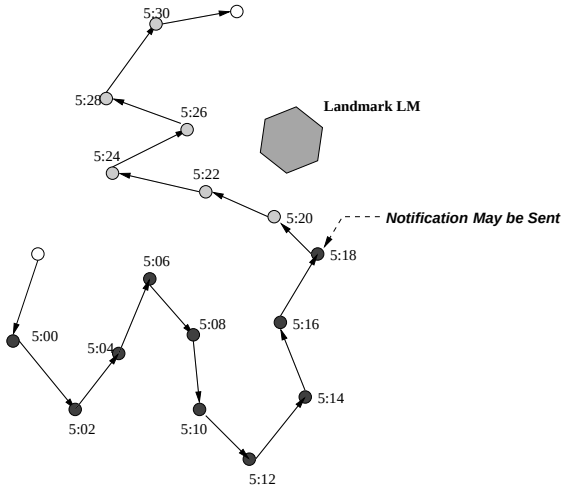

Figure 3: Notification for *Moving_Towards*

A possible scenario is illustrated in Figure 3 and we have similar observations as in Section 3.1 for the case of *moving along* predicate, except, now the semantic implications of the continuous satisfaction of the predicate for the desired interval (5 minutes) brings some other specific issues:

1. A purely query-like approach in which one would wait until 5:30 and then pose the corresponding query is, again, unacceptable. As shown in Figure 3, **RN3** is satisfied at 5:18 because between 5:12 and 5:18 the object was continuously moving towards LM for 6 minutes. Hence, the corresponding notification should have been sent at 5:18.

2. Again, one may be tempted to set up a trigger, say **TRN3**, which upon every location update (EVENT) would check if the distance between obj_2 and LM was non-increasing for an interval of 5 minutes (CONDITION) and, subsequently, notify the user (ACTION). However, aside from the questions regarding the issues of how to exactly express it, the "classical" trigger may involve unnecessary calculations for processing the **RN3**, by using the entire "current_history" of the object's motion. As a particular example, at 5:14 there is absolutely no need to query any history before 5:12.

3. A peculiarity brought by this example is that within the time-interval of interest for the request, multiple notifications may be generated. The source is two-fold:

• Although this is not illustrated in Figure 3, one can easily think of settings in which a satisfaction of the particular *moving towards* is followed by *moving away* which, in turn, is followed by another *moving towards* time-interval. This can cause multiple notifications to be sent for a given request.

• A more intriguing reason is that at 5:20 the object has completed another interval of 6 (> 5) minutes ([5:14,5:20]) during which it was continuously *moving towards* the target – landmark LM. One may object that the location updates at 5:16 and 5:18 were already used for the notification sent at 5:18 – and the objection may be either "sustained" or "over-ruled". This is due to the fact that it was not spec-

ified *what* is to be done to the primitive events that were already used throughout the history. To cope with such issues, the notification system must allow an explicit choice of a *policy* for *consuming* primitive events upon detection of the composite event. The Events Management System SNOOP [6] proposed several different policies for *consumption* of the constituent primitive events upon a detection of a composite event. Each policy specifies what happens with the set of instances of the primitive events in between (and including) the particular pair of (*initiator event*, *terminator event*). Thus, for the scenario illustrated in Figure 3, if one does not want a notification sent at 5:20 (i.e., a brand new "observation-interval" is started at 5:18), then all the updates at 5:12, 5:14 and 5:16 should be, in a sense, flushed out from consideration and a new initiator event will be sought for, with the location update at 5:18. In the parlance of the SNOOP system, this is similar to the *cumulative* consumption of the primitive constituent events. The other option is to flush out only the location update from 5:12 as a possible initiator, and use the one at 5:14 as the new initiator. This will combine with the next location update (5:20) as the new terminator event and generate another notification. This behavior corresponds to the *chronicle* consumption policy in SNOOP. Thus, when monitoring a particular request, one may have to choose the consumption policy beforehand.

3.2.1 Algorithmic Processing of the Moving_Towards Predicate

We use ideas similar to the ones in Section 3.1.1, in the sense of accumulating the time that a given line-segment s of the trajectory of obj_1 is *moving towards* the landmark LM for each location update. The algorithm which calculates the portion of the time that a given line segment $(x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1})$ has spent *moving_towards* a given polygon can be outlined as follows:

Moving_Towards(LM, (x_i, y_i, t_i), $(x_{i+1}, y_{i+1}, t_{i+1})$)

1. *Construct the Voronoi diagram $V_d(LM)$ of LM*
2. *Decompose each cell of the diagram*
3. *Shoot a ray from (x_i, y_i) towards (x_{i+1}, y_{i+1}) and obtain the intersections with $V_d(LM)$*
4. *Use linear interpolation to obtain the total time spent towards LM*

Observe that, when constructing $V_d(LM)$, the Voronoi diagram of the landmark LM [9], even if LM is bounded by line segments, the cells of $V_d(LM)$ need not be convex. We separate between the cells of the edges and those of the vertices of LM, and apply geodesic triangulation to each one of them [7, 15]. Recall (c.f. Section 3.1.1) that this is needed to answer ray shooting queries in $O(\log n)$ time.

During an update, we trace the segment s inside the diagram by shooting rays, and the calculation of the time spent moving towards the (boundary of the) LM is based on the following observations:

1. For as long as the (route of the) segment remains within the cell of an edge, it can only move towards all the time, or move away all the time. We add the length of the part of the trajectory inside the cell to the $t^{towards}$ accumulator.
2. If (part of) the segment s is inside the cell $V_d(LM)$ belonging to a vertex, it can first move towards, then away – possible critical point is at the projection of the vertex on the trajectory.

In any case, each update-segment s_i takes time proportional to the number k_i of cells it traverses (with an overhead of $O(\log n)$ due to the ray shooting query), which is again output-sensitive, since k_i is the number of times the closest feature of the landmark LM may change.

3.2.2 Behavioral Aspects of the MOD

Again we observe that the desired operational behavior corresponding to the invocations of the algorithm in Section 3.2.1 upon successive updates, cannot be declaratively specified using the existing, commercially available, ORDBMS. In order to illustrate how to achieve it in MOD settings, we will assume that we have readily available the predicate $TimeTowards(LM, (x_1, y_1, t_1), (x_2, y_2, t_2), t^{towards})$, which is true whenever $t^{towards}$ is the total time between t_1 and t_2, during which the object obj_2 was *moving towards* the target landmark LM. Assume that $(x', y', t') =$ *previous_last(obj_2)*, and and $(x, y, t) =$ *last(obj_2)*. Then, the syntactic elements of the *dynamics-aware* trigger which monitors the request **RN3** are:

TRN3dyn:

1. ON *location_update* (oid_2, x, y, t)
2. IF $TimeTowards(LM, (x, y, t), (x', y', t'), t^{towards}) \wedge$
   $(t^{towards} = t - t') \wedge (t - t_{reference} \geq 5)$
3. THEN
4. *Send_Notification* AND
   Execute the chosen consumption policy
5. ELSE
6. $t_{reference} = t$

The specification states the intended behavior for the processing of the *moving towards* predicate. The only ambiguity may arise with the value of $t_{reference}$, which is the only non-bound variable in the IF part. Initially, it is set to the beginning time of the **RN3** interval of interest – $t_{reference} = 5:00$ – and it acts, in a sense, like a global variable for detecting the desired condition (*continuously for 5 minutes*). Subsequently, it is maintained up-to-date any time the condition is invalidated by a particular location update in the MOD (for clarity of presentation, we assume that it can be only (re)set to the values of *time* of a particular *(location,time)* update). The second conjunct in line 4 *(Execute the chosen consumption policy)* pertains to the discussion about multiple notifications and it affects the value of the $t_{reference}$ variable. In particular, in case the application would like another notification sent at 5:20 (chronicle consumption), then $t_{reference}$ is set to the value of the next point of the trace which satisfied the condition in line 2 ($t_{reference} = 5:14$). On the other hand, if the application at hand does *not* want another notification sent at 5:20 (i.e. it requires a "fresh" sequence of 5 minutes during which oid_2 is continuously *moving towards* the landmark LM), then the effect of the *Execute the chosen consumption policy (cumulative one)* conjunct in line 4. would set $t_{reference} = 5:18$.

4. THE $(ECA)^2$ PARADIGM

The results in the "classical" ECA paradigm abound [22, 32] and we will not re-hash all the possible classifications of the different semantic dimensions and their respective choices in a particular Active Database System, for which a very comprehensive analysis is provided in [10].

In the $(ECA)^2$ paradigm, we allow the user to declaratively specify modifications to a given trigger, upon the failure of the *condition* part. Intuitively[3], an active rule in the *Evolving and Context-Aware Event-Condition-Action* paradigm, for the purpose of processing the dynamic predicates like the ones introduced in this paper, can be formulated as follows:

```
ON EVENT <trigger consumption>
         <composite event consumption>
   IF CONDITION
      THEN ACTION
   ELSE <parent consumption>
      MODIFY EVENT/CONDITION/ACTION
```

Similarly to the *Extended Event-Condition-Action* ($EECA$) paradigm in [10], we allow the option of a particular rule to *consume* its triggering event – `(trigger consumption)` in three ways: *locally*; *globally* (i.e., no other rule can be triggered by the same event); and *no-consumption* (i.e., the event can re-trigger the same rule). However, in addition to $EECA$, we also consider composite events that can trigger a particular rule and the choice of policy for consuming the primitive constituent events, which can occur either upon the detection of the triggering event, or upon the condition evaluation, or upon the execution of the action part of the trigger (not shown in the above syntax). In this work we do not explicitly consider different *coupling modes* between the detection of the event; evaluation of the condition; and execution of the action, which can be *immediate*, *deferred* and *detached* (in a separate transaction) [10]. The main idea of the $(ECA)^2$ is to allow the users to specify which particular evolution (among possible ones) is of interest, and what is to be monitored. In Section 3 we presented the triggers **TRN2dyn** and **TRN3dyn** used in the reactive management for the predicates *moving along* and *moving towards*, respectively, which illustrated the evolution of the *condition* part.

Due to lack of space, we cannot elaborate formally all the aspects of the $(ECA)^2$ (c.f. [28]), but we would like to explain the semantics of one particular syntactic element of $(ECA)^2$ rules that we did not address so far. The value of the `<parent consumption>` can be either *yes* or *no*, and to explain its role, consider the following request:
RN4: *"Notify me when an object is moving towards the landmark* LM *continuously for 5 minutes between 5:00 and 5:30", if I have less then 5 armored vehicles in the Base B1".*
Even though obj_x may be moving continuously towards $B1$, for as long as there are ≥ 5 armored vehicles there, the user will receive no notification. However, once it has been detected that obj_x is *moving towards* $B1$, the user may be interested in monitoring *another* criterion:
RN4': *"Subsequently, notify me when that object is closer than 10 miles to B1".*
This is an example in which, once certain event is detected but the condition fails, a new trigger is spawned (`MODIFY` the `EVENT`) and the original (parent) trigger *ceases* to exist: `parent consumption` = *yes*. On the other hand, the user may be interested in continuing to monitor the original criterion along the development of (the model of) the battlefield, *together* with the new criterion, in which case:
RN4": *"Subsequently, also notify me when that object is closer than 10 miles to B1".*
Thus, in the case of **RN4"**, the *parent* trigger will *continue* to exist along with the newly created trigger (`parent consumption` = *no*) which, again, was generated due to the failure of the condition part of the *parent* when its event was detected.

We would like to point out that one of the main motivations behind the $(ECA)^2$ paradigm was enabling reactive behavior in heterogeneous environments. In this work we focused on MOD settings and notification services, however, one may very well observe that even in the scenarios for **RN2** and **RN3**, introducing the *moving along* and *moving towards* predicates, it may be the case that the *(location,time)* information will be actually detected by a set of sensors deployed in a given area (it is unlikely that an enemy unit will voluntarily send GPS updates).

5. RELATED WORK

Due to the specific nature of the spatio-temporal domain, the abundance of works in active databases (e.g., [22, 32]) cannot be applied directly to the MOD settings, although, as we indicated, they provide a solid foundation for extension of the management of reactive behavior. In particular, the *Extended Event-Condition-Action (EECA)* model in [10] took a step towards more "event-aware" active rules. Although it utilizes the notion of querying over the history of the evolution based on specification of the proper events, the work did not have the full concept of composite events (i.e., it only considers a disjunction of primitive events). None of the works addressed the problem of evolution of the triggers in a context-awareness manner.

MOD research has brought many interesting results and the recent collection [18] provides an extensive list of references. However, none of these works has explicitly addressed the problem of managing continuous requests in a reactive manner. In particular, [26] introduced the category of *persistent* queries. Since, in our paradigm, the condition evaluation is *context-aware* about the impact of the changes of the *(location,time)* updates on the overall content and we also consider different event consumption policies, we believe that our work provides a good framework for managing. *persistent* queries.

Efficient maintenance of the answers to continuous queries in the settings where the *Motion_Plan* of the objects is obtained by a sequence of location updates in time is addressed in [20]. Although we did not address the issues of scalability, our work is complementary to [20], in the sense of evolution of conditions and/or events.

Many works on Event-based and Context-Aware Notification systems: the GUIDE project [8], TIP system [16], CATIS system [21] (to name a few), are concerned with the filter-

[3]See [28] for a detailed domain description and declarative semantics.

ing, formating and delivering the desired information to a mobile user based on his current location, preferences and communication device. However, these works mostly match *static* conditions with the dynamically changing information of the user's location.

The CAT system [27, 29] addressed the problem of managing continuous spatio-temporal range queries using triggers. Orthogonally to this work, the proposed framework assumed that the *Motion_Plan* was represented as a complete future trajectory and the main focus of the context-awareness was to correctly identify and update the trajectories that may be affected by abnormal traffic conditions, and update the answer sets of the pending queries. In this work we focused on dynamic topological predicates and on the aspects of the reactive behavior which enable the MOD users a more flexible specification of the triggers, for the purpose of an overall optimization of the processing of the predicates.

6. CONCLUSIONS AND FUTURE WORK

We introduced two *dynamic topological* predicates used in Requests for Notification and addressed their efficient processing from two perspectives: *algorithmic* and *reactive behavior* of MOD. We demonstrated that for some requests, besides the dynamics of the *(location,time)* information of the objects, its impact on (the dynamics of) other values in a given state may have to be taken into consideration, along with the evolution which brought the system to that particular state. To address this, we argued that the "classical" ECA paradigm may not be well-suited for specifying the reactive behavior in MOD settings and we proposed the $(ECA)^2$ paradigm.

Part of our ongoing work is considering the impact of the different models (e.g., *(location, time, velocity)* updates [26] or *full-future trajectory*, based on electronic maps [30]) on the types of requests and their processing, by developing a comprehensive type system and carefully identifying the set of operators and their processing for various argument signatures, similarly to [12]. Since in our paradigm the *condition* is, in a sense, context-aware about the impact of the changes of the *(location,time)* updates (event generation) on the overall content, and we also consider different consumption policies for composite events, we believe that our work provides a good foundation for managing *persistent* queries [26].

Novel paradigms need to be "brought to life" in an actual implementation, and a possible approach is to implement this type of reactive behavior anew, over a platform with a rich type system e.g., SECONDO (surveyed in [3]). Along the terminology used in the recent survey [3] and based on our experiences with the CAT system [29], we believe that the paradigm which we proposed in this work can be implemented on top of an extensible ORDBMS, which is the goal of our CAT++ project [28]. This has advantages, to say the least, of pushing the processing of the rule as much as possible into the underlying ORDBMS (c.f. [31]). Along this work, we are also investigating the limits that the underlying ORDBMS may impose on the expressiveness of the $(ECA)^2$ paradigm.

Acknowledgments: The authors are grateful to the anonymous reviewers for their comments and suggestions.

7. REFERENCES

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *Handheld and Ubiquitous Computing: First International Symposium*, 1999.

[2] P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of minkowski sums. *Computational Geometry*, 21(1-2), 2002.

[3] M. Breunig, C. Türker, M. Böhlen, S. Dieker, R.H. Güting, C. Jensen, L. Relly, P. Rigaux, H.-J. Schek, and M. Scholl. Architectures and implementations of spatio-temporal database management systems. In *Spatio-Temporal Databases – the Chorochronos Approach*. 2003.

[4] F. S. Brundick and G. W. Hartwig. Model – based situational awareness. In *Proceedings of the Joint Service Combat Identification Systems Conference CISC-97*, April 1997.

[5] A. Chakravarthy and D. Ghose. Capturability of realistic generalized true proportional navigation. *IEEE Transactions on Aerospace and Electronic Systems*, 32(1), 1996.

[6] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.K. Kim. Composite events for active databases: Semantics, contexts and detection. In *20th VLDB Conference*, 1994.

[7] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12, 1994.

[8] K. Cheverst, N. Davies, K. Mitchell, and A. Friday. Experiences of developing and deploying a contex-aware tourist guide: The guide project. In *MOBICOM*, 2000.

[9] M. de Berg, M. Overmars, M. van Kreveld, and O. Schwartzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2 edition, 2000.

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

[11] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2), 1998.

[12] R. H. Güting, M. H. Bohlen, M. Erwig, C. S. Jensen, N. Lorentzos, E. Nardeli, M. Schneider, and J. R. R. Viqueira. Spatio-temporal models and languages: An approach based on data types. In *Spatio-Temporal Databases – the Chorochronos Approach*. 2003.

[13] R.H. Güting and M. Schneider. Realm-based spatial data types: The rose algebra. *VLDB Journal*, 4, 1995.

[14] A. Gutman. R-trees: A dynamic indexing structure for spatial searching. In *ACM SIGMOD*, 1984.

[15] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *Journal of Algorithms*, 18, 1995.

[16] A. Hinze and A. Voisard. Location-and time-based information delivery in tourism. In *SSTD*, 2003.

[17] W. Kainz, M. Egenhofer, and I. Greasley. Modeling spatial relations and operations with partially ordered sets. *International Journal of Geographical Information Systems*, 7(3), 1993.

[18] M. Koubarakis, T. Sellis, A.U. Frank, S. Grumbach, R.H. Güting, C.S. Jensen, N. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.-J. Scheck, M. Scholl, B. Theodoulidis, and N. Tryfona, editors. *Spatio-Temporal Databases – the CHOROCHRONOS Approach.* Springer-Verlag, 2003.

[19] U. Meissen, S. Pfennigschmidt, A. Voisard, and T. Wahnfried. Context- and situation-awareness in information logistics. In *International EDBT Workshop on Pervasive Information Management (PIM)*, 2004.

[20] M.F. Mokbel, X. Xiong, and W.G. Aref. Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *ACM SIGMOD Internation Conference on Management of Data*, 2004.

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

[22] N. W. Paton. *Active Rules in Database Systems.* Springer-Verlag, 1999. New York.

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

[24] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Workshop on Mobile Computing Systems and Applications*, 1994.

[25] J. Schiller and A. Voisard. *Location-based Services.* Morgan Kaufmann Publishers, 2004.

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

[27] G. Trajcevski and P. Scheuermann. Reactive maintenance of continuous queries. *ACM SIGMOBILE Mobile Computing and Communications Review*, 8, 2004.

[28] G. Trajcevski, P. Scheuermann, H. Brönnimann, and A. Voisard. Dynamic content and context-aware reactive behavior in moving objects databases. Technical Report TR-25-04, Department of ECE, Northwestern University, 2004.

[29] G. Trajcevski, P. Scheuermann, O. Wolfson, and N. Nedungadi. Cat: Consistent answers to continuous queries using triggers. In *International Conference on Extending Database Technology (EDBT)*, 2004.

[30] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM Transactions on Database Systems*, 29(3), 2004.

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

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