# Moving Objects Databases: Issues and Solutions

Ouri Wolfson[†] Bo Xu[‡] Sam Chamberlain[§] Liqin Jiang[¶]

## Abstract

*Consider a database that represents information about moving objects and their location. For example, for a database representing the location of taxi-cabs a typical query may be: retrieve the free cabs that are currently within 1 mile of 33 N. Michigan Ave., Chicago (to pick-up a customer). In the military, moving objects database applications arise in the context of the digital battlefield, and in the civilian industry they arise in transportation systems.*

*Currently, moving objects database applications are being developed in an ad hoc fashion. Database Management System (DBMS) technology provides a potential foundation upon which to develop these applications, however, DBMS's are currently not used for this purpose. The reason is that there is a critical set of capabilities that are needed by moving objects database applications and are lacking in existing DBMS's. The objective of our Databases fOr MovINg Objects (DOMINO) project is to build an envelope containing these capabilities on top of existing DBMS's. In this paper we describe the problems and our proposed solutions.*

## 1 INTRODUCTION

Consider a database that represents information about moving objects and their location. For example, for a database representing the location of taxi-cabs a typical query may be: retrieve the free cabs that are currently within 1 mile of 33 N. Michigan Ave., Chicago (to pick-up a customer); or for a trucking company database a typical query may be: retrieve the trucks that are currently within 1 mile of truck ABT312 (which needs assistance); or for a database representing the current location of objects in a battlefield a typical query may be: retrieve the friendly helicopters that are in a given region, or, retrieve the friendly helicopters that are expected to enter the region within the next 10 minutes. The queries may originate from the moving objects, or from stationary users. We will refer to applications with the above characteristics as moving-objects-database (MOD) applications, and to queries as the ones mentioned above as MOD queries.

In the military, MOD applications arise in the context of the digital battlefield (see [5, 6]), and in the civilian industry they arise in transportation systems. For example, Omnitracs developed by Qualcomm (see [26]) is a commercial system used by the transportation industry, which enables MOD functionality. It provides location management by connecting vehicles (e.g. trucks), via satellites, to company databases. The vehicles are equipped with a Global Positioning System (GPS), and they automatically and periodically report their location.

Currently, MOD applications are being developed in an ad hoc fashion. Database Management System (DBMS) technology provides a potential foundation upon which to develop MOD applications, however, DBMS's are currently not used for this purpose. The reason is that there is a critical set of capabilities that are needed by MOD applications and are lacking in existing DBMS's. The following is a discussion of the needed capabilities.

- **Location Modeling**

  Existing DBMS's are not well equipped to handle continuously changing data, such as the location of moving objects. The reason for this is that in databases, data is assumed to be constant unless it is explicitly modified. For example, if the salary field is 30K, then this salary is assumed to hold (i.e. 30K is returned in response to queries) until explicitly updated. Thus, in order to represent moving objects (e.g. vehicles) in a database and answer queries about their location, the vehicle's location has to be continuously updated. This is unsatisfactory since either the location is updated very frequently (which would impose a serious performance overhead), or, the answer to queries is outdated. Furthermore, assuming that the location updates are generated by the moving objects themselves and transmitted via wireless networks, frequent updating would also impose a serious wireless bandwidth overhead.

- **Lingustics Issues**

  Generally, a query in MOD applications involves spa-

*This research was supported in part by Army Research Labs grant DAAL01-96-2-0003, NSF grant IRI-9408750, DARPA grant N66001-97-2-8901, NATO grant CRG-960648, and a Hughes Research Labs gift

†Department of Electrical Engineering and Computer Science, University of Illinois, Chicago, IL 60607, wolfson@eecs.uic.edu, 312-996-6770, 312-413-0024(fax)

‡Department of Electrical Engineering and Computer Science, University of Illinois, Chicago, IL 60607, bxu@eecs.uic.edu

§Army Research Laboratory, Aberdeen Proving Ground, MD

¶Department of Electrical Engineering and Computer Science, University of Illinois, Chicago, IL 60607,ljiang1@eecs.uic.edu

tial objects (e.g. points, lines, regions, polygons) and temporal constraints. Consider for example the query: "Retrieve the objects that will intersect the polygon P within the next 3 minutes". This is a spatial and temporal range query. The spatial range is the polygon P, and the temporal range is the time interval between now and 3 minutes from now. Similarly, there are spatio-temporal join queries such as: "Retrieve the pairs of friendly and enemy aircraft that will come within 10 miles of each other, and the time when this will happen." Traditional query languages such as SQL are inadequate for expressing such queries. Although spatial and temporal languages have been studied in the database research community, the two types of languages have been studied independently, whereas for MOD databases they have to be integrated. Furthermore, spatial and temporal languages have been developed for data models that are inappropriate for MOD applications (due, for example, to the modeling problem mentioned above).

- **Indexing**

  Observe that the number of moving objects in the database may be very large (e.g., in big cities with millions of inhabitants). Thus, for performance considerations, in answering MOD queries we would like to avoid examining the location of each moving object in the database. In other words, we would like to index the location attribute. The problem with a straightforward use of spatial indexing for this purpose is that the continuous change of the locations implies that the spatial index has to be continuously updated. This is clearly an unacceptable solution.

- **Uncertainty/Imprecision**

  The location of a moving object is inherently imprecise because, regardless of the policy used to update the database location of the object (i.e. the object-location stored in the database), the database location cannot always be identical to the actual location of the object. This inherent uncertainty has various implications for database modeling, querying, and indexing. For example, for range queries there can be two different kinds of answers, i.e. the set of objects that "may" satisfy the query, and the set that "must" satisfy the query. Thus, different semantics should be provided for queries. Another approach would be to compute the probability that an object satisfies the query. Although uncertainty in databases has been studied extensively, the new modeling and spatio-temporal capabilities needed for moving objects introduce the need to revisit existing solutions.

  Additionally, existing approaches to deal with uncertainty assume that some uncertainty information is associated with the raw data stored in the database. How is this initial uncertainty obtained? For MOD applications the question becomes how to quantify the location uncertainty? How to quantify the trade-off between the updating overhead and the uncertainty/imprecision penalty, and how frequently should a moving object update its location. How to handle the possibility that a moving object becomes disconnected and cannot send location updates?

Therefore, there is a critical set of capabilities that have to be integrated, adapted, and built on top of existing DBMS's in order to support moving objects databases. The objective of our Databases fOr MovINg Objects (DOMINO) project is to build an envelope containing these capabilities on top of existing DBMS's. The key features of our approach are the following.

- **Dynamic Attributes**

  In our opinion, the key to overcoming the location modeling problem is to enable the DBMS to *predict* the future location of a moving object. Thus, when the moving object updates the database, it provides not only its current location, but its expected future locations. For example, if the DBMS knows the speed and the route of a moving object, then it can compute its location at any point in time without additional updates.

  Thus, we propose a data model called the Moving Objects Spatio-Temporal (or MOST for short) model. Its novelty is the concept of a dynamic attribute, i.e. an attribute whose value changes continuously as time progresses, without being explicitly updated. So, for example, the location of a vehicle is given by its dynamic attribute which consists of motion information (e.g., north on route 481, at 60 miles/hour). In other words, we devise a higher level of data abstraction where an object's motion information (rather than its location) is represented as an attribute of the object. Obviously the motion information of an object can change (thus the dynamic attribute needs to be updated), but in most cases it does so less frequently than the location of the object. We devised mechanisms to incorporate dynamic attributes in existing data models and capabilities to be added to existing query processing systems to deal with dynamic attributes.

- **Spatial and Temporal Query Language**

  We introduced a temporal query language called Future Temporal Logic (FTL) for query and trigger specifications in moving objects databases. The language is natural and intuitive to use in formulating MOD queries, and it uses both spatial operators (e.g. object INSIDE polygon) and temporal operators (e.g. UNTIL, EVENTUALLY in the future). We are developing algorithms for processing FTL queries on databases with dynamic attributes. We have implemented FTL in a prototype running on top of Sybase and on top of MS Access.

- **Indexing Dynamic Attributes**

We propose the following paradigm for indexing dynamic attributes. The indexing problem is decomposed into two sub-problems; first is the geometric representation of a dynamic attribute value (i.e. a moving object's speed, initial location, and starting time) in multidimensional time-space, and second is the spatial indexing of the geometric representation. The geometric representation subproblem concerns the question: how to construct the multidimensional space, and how to map an object (more precisely, a dynamic attribute value) into a region (or a line, or a point) in that space, and how to map a query into another region in that space, so that the result of the query are the objects whose regions intersect the query region. The object region is updated only when the dynamic attribute is explicitly updated (e.g. when the speed of the object changes) rather than continuously. The spatial indexing subproblem concerns the question how to find the intersection-of-regions mentioned above in an efficient way. The latter subproblem can be solved by an existing spatial indexing method, but it is an open problem which method is most appropriate for a particular geometric representation and dynamic attribute values distribution. We have devised several solutions to the geometric representation subproblem, and in this paper we present two of them, namely the *value-time space* representation and the *intercept-slope space* representation.

- **Uncertainty/Imprecision Management**

  We extended our data model, query language, and indexing method to address the uncertainty problem. The data model was extended by enabling the provision of an uncertainty interval in the dynamic attribute. More specifically, at any point in time the location of a moving object is a point in some uncertainty interval, and this interval is computable by the DBMS. Thus, the DBMS replies to a query requesting the location of a moving object $m$ with the following answer A: "$m$ is on route 698 at location ($x$,$y$), with an error (or deviation) of at most 2 miles". The bound $b$ on the deviation (2 miles in the above answer) is provided by the moving object, i.e. the object commits to send a location update when the deviation reaches the bound. The FTL language is also extended. Two kinds of semantics, namely *may* and *must* semantics, are incorporated, and the processing algorithms are adapted for these semantics. The indexing method is also extended to enable the retrieval of both, moving objects that "must be" in a particular region, and moving objects that "may be" in it.

  We also addressed the question of determining the uncertainty associated with a dynamic attribute, i.e. the bound $b$ mentioned above. We proposed a cost based approach which captures the tradeoff between the update overhead and the imprecision. The location imprecision encompasses two related but different concepts, namely deviation and uncertainty. The deviation of a moving object $m$ at a particular point in time $t$ is the distance between $m$'s actual location at time $t$, and its database location at time $t$. For the answer $A$ above, the deviation is the distance between the actual location of $m$ and ($x$,$y$). On the other hand, the uncertainty of a moving object $m$ at a particular point in time $t$ is the size of the interval in which the object can possibly be. For the answer $A$ above, the uncertainty is 4 miles. The deviation has a cost (or penalty) in terms of incorrect decision making, and so does the uncertainty. The deviation (uncertainty) cost is proportional to the size of the deviation (uncertainty). The tradeoff between imprecision and update overhead is captured by the relative costs of an uncertainty-unit, a deviation-unit, and an update-overhead unit. Using the cost model we propose update policies that establish the uncertainty bound $b$ in a way that minimizes the expected total cost. Furthermore, we propose an update policy that detects disconnection of the moving object at no additional cost.

- **Simulation Testbed**

  We are building a simulation testbed in which the performance of a moving objects database application can be evaluated. The input to the simulation system is a set of moving objects, their trajectories, their speed variations over time, the costs of deviation, the cost of uncertainty, the cost of communication, the wireless bandwidth distribution over the geographic area, and the location update policy used by each moving object. The objective is to determine the performance of MOD queries, as well as to answer questions such as: How many objects can be supported for an average imprecision that is bounded by $x$, and a wireless bandwidth allocated to location updates that is bounded by $y$? Or, given $n$ moving objects and a bound of 10% on the imprecision, what percentage of the bandwidth is used for location updates?

The rest of this paper is organized as follows. In section 2 we present the MOST data model. In section 3 we discuss the FTL query language. In section 4 we discuss the indexing of dynamic attributes. In section 5 we discuss our extensions to the above solutions to address the uncertainty problem. We also propose three update policies and we discuss the results of their comparison by simulation. In section 6 we present the prototype implementation. In section 7 we discuss relevant work, and in section 8 we discuss future work.

## 2 THE MOST DATA MODEL

In traditional DBMS's, data is assumed to be constant unless it is explicitly modified. Thus, in order to represent moving objects (e.g. cars) in a database, and answer queries about their location (e.g., How far is the car with license plate RWW860 from the nearest hospital?) the car's location has to be continuously updated. This is unacceptable since either the location is updated very frequently (which would impose a serious performance and wireless-bandwidth overhead), or, the answer to queries is outdated. Furthermore, it is possible that due to disconnection an object cannot continuously update its location.

We propose to solve the continuously changing location problem by representing the location as a function of time; it changes as time passes, even without an explicit update. So, for example, the location of a helicopter is given as a function of its motion vector (e.g., north, at 60 miles/hour). In other words, we consider a higher level of data abstraction, where an object's motion vector is represented as an attribute of the object. Obviously, the motion vector of an object can change, but in most cases it does so less frequently than the location of the object.

We propose a data model called Moving Objects Spatio-Temporal (or MOST for short). Its main contribution is the concept of *dynamic attributes*, i.e. attributes that change continuously as a function of time, without being explicitly updated. In other words, the answer to a query depends not only on the database contents, but also on the time at which the query is entered. In contrast, a *static attribute* of an object is an attribute in the traditional sense, i.e. it changes only when an explicit update of the database occurs.

Formally, a *dynamic attribute* $A$ is represented by three sub-attributes, $A.updatevalue$, $A.updatetime$, and $A.function$, where $A.function$ is a function of a single variable $t$ that has value 0 at $t = 0$. The *value* of a dynamic attribute depends on the time, and it is defined as follows. At time $A.updatetime$ the value of $A$ is $A.updatevalue$, and until the next update of $A$ the value of $A$ at time $A.updatetime + t_0$ (where $t_0$ is a positive number) is given by $A.updatevalue + A.function(t_0)$. An explicit update of a dynamic attribute may change its value sub-attribute, or its function sub-attribute, or both sub-attributes.

In this paper we are concerned with dynamic attributes that represent spatial coordinates, but the model can be used for other hybrid systems, in which dynamic attributes represent, for example, temperature, or fuel consumption.

For a moving object, we can model its *location attribute* $L$ by two dynamic attributes $L.x$, and $L.y$, each with its own update value, function, and update time, representing the $x$ and $y$ coordinates of the object respectively (all our concepts and results can be extended to motion in three-dimensional space). The object updates its location when its speed changes. This is straight-forward for objects that move freely in space (e.g. aircraft). However, this would be inefficient (i.e. may generate many updates) for objects moving along a winding route, since each turn would constitute a change of $L.x.function$ and $L.y.function$.

To address this problem, we can extend the dynamic attribute concept to include the route as follows. The location attribute is a dynamic attribute with five sub-attributes, namely $L.route$, $L.x.updatevalue$, $L.y.updatevalue$, $L.updatetime$, and $L.speed$. Among them, $L.route$ is (the pointer to) a line spatial object indicating the route on which an object is moving. $L.x.updatevalue$ and $L.y.updatevalue$ are the $x$ and $y$ coordinates of a point on $L.route$; it is the location of the moving object at time $L.updatetime$, i.e. the time of the last location-update. $L.speed$ is a linear function of the form $f(t) = b \cdot t$. It is defined by the speed $b$ of the moving object, and it gives the current distance from the starting location as a function of the time $t$ elapsed since $L.updatetime$. The location at time $L.updatetime + t$ is the point ($x$,$y$) which is at route-distance [1] $L.speed \cdot t$ from the point with coordinates ($L.x.updatevalue$, $L.y.updatevalue$).

## 3 THE FTL LANGUAGE

A nontemporal query is a query that pertains to the present time, e.g. "Retrieve all the objects that are currently inside the polygon P". A regular query language such as SQL or OQL augmented with spatial predicates can be used for nontemporal queries on moving objects. Now consider for example the following temporal query $Q$: "Retrieve the pairs of objects $o$ and $n$ such that the distance between $o$ and $n$ stays within 5 miles until they both enter the polygon $P$". Expressing such a temporal query would be cumbersome in SQL or OQL. Assume that for each predicate $G$ there are functions $begin\_time(G)$ and $end\_time(G)$ that give the beginning and ending times of the first time-interval during which $G$ is satisfied; also assume that "now" denotes the current time. Then the query $Q$ would be expressed as follows in SQL or OQL.

RETRIEVE o,n
FROM Moving-Objects
WHERE begin_time($DIST(o, n) \leq 5$) $\leq$ now
    and end_time($DIST(o, n) \leq 5$) $\geq$
    begin_time($INSIDE(o, P) \wedge INSIDE(n, P)$).

where $DIST(o, n)$ and $INSIDE(o, P)$ are both spatial methods. $DIST(o, n)$ returns the distance between $o$ and $n$, while $INSIDE(o, P)$ indicates whether or not $o$ is inside $P$.

The FTL query language enables a natural specification of future queries, i.e. queries pertaining to the **future** states of the system being modeled. Since the language and system are designed to be installed on top of an existing DBMS, the FTL language assumes an underlying nontemporal query language provided by the DBMS. However, the FTL language is not dependent on a specific underlying query language, or, in other words, can be installed on top of any DBMS.

The formulas (i.e. queries) of FTL use two basic future temporal operators Until and Nexttime . A formula of the form $f$ Until $g$ is satisfied at a state, if and only if one of the following two cases holds: either $g$ is satisfied at that state, or there exists a future state in the history where

[1] the route-distance between two points on a give route is the distance along the route between the two points. We assume that it is straightforward to compute the route-distance between two points, and the point at a given route-distance from another point.

$g$ is satisfied and until then $f$ continues to be satisfied. A formula of the form $\mathsf{Nexttime}\ f$ is satisfied at a state, if and only if the formula $f$ is satisfied at the next state of the history.

In FTL, the query Q above can be expressed as follows:

RETRIEVE o,n
WHERE $DIST(o,n) \leq 5$
Until $(INSIDE(o,P) \wedge INSIDE(n,P))$

Other temporal operators, such as $\mathsf{Eventually}\ f$ and $\mathsf{Always}\ f$ can be expressed in terms of the above two basic operators. The temporal operator $\mathsf{Eventually}\ f$ asserts that $f$ is satisfied at some future state, and it can be defined as $true\ \mathsf{Until}\ f$. The temporal operator $\mathsf{Always}\ f$ asserts that $f$ is satisfied at all future states, including the present state, and it can be defined as $\neg\ \mathsf{Eventually}\ \neg f$.

The FTL language also uses the following bounded temporal operators that pertain to real-time:

- **Eventually_within_c** $(g)$ asserts that the formula $g$ will be satisfied within the next $c$ time units.
- **Eventually_after_c** $(g)$ asserts that $g$ holds after at least $c$ units of time.
- **Always_for_c** $(g)$ asserts that the formula holds continuously for the next $c$ units of time.
- ($g$ **until_within_c** $h$) asserts that there exists a future instance within $c$ units of time where $h$ holds, and until then $g$ continues to be satisfied.

In our system, a query is specified by the following syntax:

RETRIEVE <target-list>
WHERE <condition>.

Here <condition> is given by a FTL formula.

For example, the following query retrieves all the objects $o$ that enter the polygon $P$ within three units of time, and have the attribute $PRICE \leq 100$.

(I) RETRIEVE o
WHERE $o.PRICE \leq 100 \wedge$
**Eventually_within_3** $INSIDE(o,P)$

The following query retrieves all the objects $o$ that enter the polygon $P$ within three units of time, and stay in $P$ for another 2 units of time.

(II) RETRIEVE o
WHERE **Eventually_within_3** $(INSIDE(o,P) \wedge$
**Always_for_2** $INSIDE(o,P))$

The following query retrieves all the objects $o$ that enter the polygon $P$ within three units of time, stay in $P$ for two units of time, and after at least five units of time enter another polygon $Q$.

(III) RETRIEVE o
WHERE **Eventually_within_3**
[ $INSIDE(o,P) \wedge$ **Always_for_2** $(INSIDE(o,P)$
$\wedge$ **Eventually_after_5** $INSIDE(o,Q))$]

We also developed an algorithm for evaluating FTL queries in the MOST model. Due to space limitations, a detailed description of the algorithm is omitted here. See [31] for a complete presentation.

# 4 INDEXING DYNAMIC ATTRIBUTES

In this section we address the issue of indexing dynamic attributes. The objective is to enable answering range queries of the form "Retrieve the objects that are currently inside the polygon P", or "Retrieve the objects whose dynamic attribute value is in the range $[a_b...a_e]$ at time $t$" (obviously without examining all the objects). The problem with a straight-forward use of spatial indexing is that since objects are continuously changing their locations, the spatial index has to be continuously updated; clearly an unacceptable solution.

We identified the following paradigm for a solution. The indexing problem can be decomposed into two subproblems, namely geometric representation of moving objects and indexing of the geometric representation. The geometric representation concerns the following question: how to construct a space (we will call it the *representation space*), and map each moving object and each query into a region (or a line, or a point) in that space, such that the result of the query is the set of all objects whose region intersects the query region. The sub-problem of indexing the geometric representation addresses the question how to find the result of the intersection in an efficient way. This sub-problem can probably be solved efficiently by one of the many existing spatial access methods (see [27] for a survey).

So far we have mainly addressed the first sub-problem. In this paper we will discuss two representations, namely the *value-time representation space* and the *intercept-slope representation space*.

- **Value-time Representation Space**

  This method plots all the functions representing the way a dynamic attribute changes with time. Thus, the representation space of this method is constructed by the x-axis representing time, and the y-axis representing the value of the dynamic attribute. An object is mapped to a trajectory that plots the location as a function of time. A range query of the form Q="Retrieve the objects whose attributes value is in the range $[a_b...a_e]$ at time $t$" is a vertical line segment, the end points of which are ($t$, $a_b$) and ($t$, $a_e$) (see Figure 1(a)). In this way, the answer set consists of all the objects that have trajectories that intersect the query line segment.

- **Intercept-slope Representation Space**

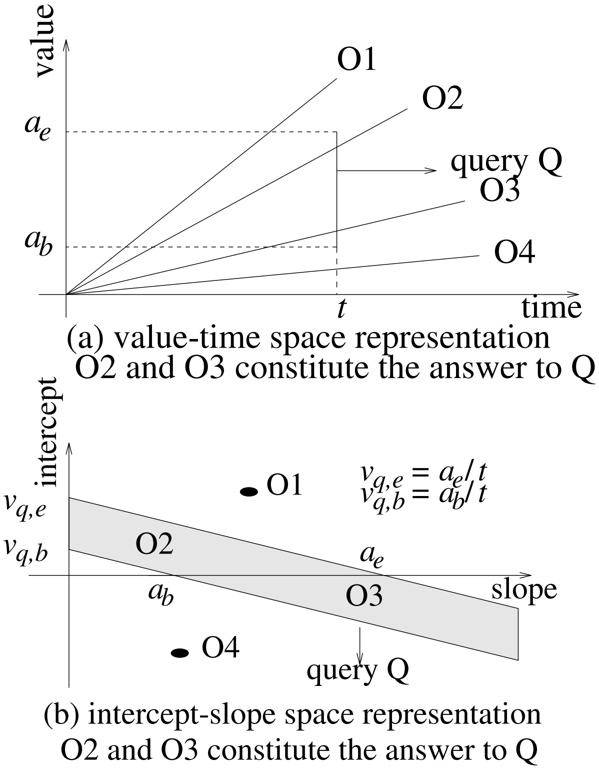


**Figure 1. Geometric representations for a range query.**

Consider an object $o$ whose location as a function of time is $f(t)$=$a$+$vt$. $a$ is called the $intercept$ and $v$ is called the $slope$. Then, the representation space is constructed by the x-axis representing the $intercept$ and the y-axis representing the $slope$. Thus the object $o$ is mapped to the point ($a$,$v$) in that space. The range query Q above is a parallelogram in the representation space (see Figure 1(b)). In this way, the answer set consists of all the objects represented by the points inside this parallelogram.

Observe that in each one of the above methods, the representation of an object in space is updated when and only when one of the sub-attributes of the location dynamic attribute is explicitly updated. It can be argued that the first method above is more efficient for querying and less efficient for updating, whereas in the second method the opposite is true. For space considerations, we omit the discussion of this claim.

Each one of the above methods works for freely moving objects. It is an open issue to find efficient geometric representations for objects that move on routes.

# 5 UNCERTAINTY MANAGEMENT

The location of a moving object is inherently imprecise because, regardless of the policy used to update the database location of a moving object (i.e. the object's location stored in the database), the database location cannot always be identical to the actual location of the object. This uncertainty has various implications for database modeling, querying, and indexing.

In this section we first extend our MOST data model to represent the uncertainty of database location (subsection 5.1), then we adapt our FTL language (5.2) and indexing method (5.3) to process "may" and "must" queries. In subsection 5.4 we discuss a cost based approach to determine when to update the location.

## 5.1 Data Modeling

In order to model the uncertainty of the database location, we first define the deviation concept. In general, the $deviation$ of the value of a dynamic attribute at a particular point in time $t$ is the difference between the actual value at time $t$, and the database value (i.e. the value stored in the database) at time $t$.

One way of modeling the uncertainty is to provide a bound on the deviation. At any point in time the moving object and the DBMS know this bound, and the moving object commits to send an update when the deviation reaches the bound. Thus, if the bound is 1 mile, then the DBMS will answer a query "what is the current location of $m$?" by an answer $A$: "the current location is $(x, y)$ with a deviation of at most 1 mile". For this answer, the uncertainty is the area of a circle with radius 1 mile. Observe that for a freely moving object, the uncertainty is the area of a circle with radius 1 mile around $(x, y)$, and for an object moving on a route, the uncertainty is an interval on the route from the point at 1 mile behind $(x, y)$ to the point at 1 mile ahead of $(x, y)$. The bound on the deviation is given by an additional sub-attribute called $L.uncertainty$.

Observe that the proposed method cannot model a bound on the deviation in the speed of a moving object (e.g. the speed is between 50 and 60 miles/hour). Similarly, the method cannot model a constraint that indicates that a moving object does not go backwards (because, as long as the object is in the uncertainty interval, its location at time $t$ can be behind its location at time $t-1$). Both problems can be addressed by an extension of the above model, but we will omit this discussion from the present paper.

## 5.2 Query Language

Consider the query Q="Retrieve the objects that are inside the polygon $P$". Because of the uncertainty in the database location, there can be two different kinds of semantics to this query, namely $may$ and $must$. Under the "may" semantics, the answer is the set of all objects that are possibly inside $P$, i.e. the objects whose uncertainty interval intersects $P$. Under the "must" semantics, this will be the set of all objects which are definitely inside $P$, i.e. the objects whose uncertainty intervals are entirely inside $P$ (see Figure 2). We have incorporated $may$ and $must$ semantics into the FTL language and the query processing algorithm.

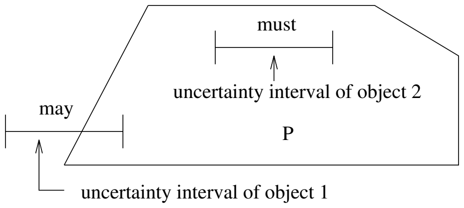


**Figure 2.** $may$ **and** $must$ **semantics**

A more general way of dealing with the uncertainty problem is to associate probabilities with answers to queries. Thus, for example, an answer to query Q would say that object 1 is inside P with probability 0.4 and object 2 is inside P with probability 1.

Observe that for queries that pertain to a future time, both $must$ and $may$ queries are tentative in the following sense. Consider the query "Retrieve all the airplanes that will come within 30 miles of the airport in the next 10 minutes". Suppose that the answer to the query Q contains airplane $a$. It is possible that after the answer is presented to the user, the motion vector of $a$ changes in a way that steers $a$ away from the airport, and the database is updated to reflect this change. Thus $a$ does not come within 30 miles of the airport in the next 10 minutes. Therefore, in this sense the answer to future queries is tentative, i.e. it should be regarded as correct according what is *currently* known about the real world, but this knowledge (e.g. the motion vector) can change.

## 5.3 Indexing

Since the semantics of queries are enriched, indexing should also be extended to efficiently process the queries of the form $Q_1$="Retrieve the objects which $must$ be inside the polygon P at time $t$" or $Q_2$="Retrieve the objects which $may$ be inside the polygon P at time $t$". In this subsection we discuss an extension of the value-time representation space (see section 4) to deal with uncertainty. We construct a plane, called the $o$-plane, to represent the location attribute of a moving object $o$. The $o$-plane is the set of uncertainty intervals of $o$, one uncertainty interval for each time unit $t \geq 0$ (see Figure 3). Thus, instead of being represented by a line (or a trajectory), an object is represented by a plane. In other words, at time $t$, the value of the location attribute is an interval instead of a point. A range query is still a line segment as before. The answer to query $Q_1$ above is the set of objects whose uncertainty intervals at time $t$ lie inside $P$ in their entirety. The answer to query $Q_2$ above is the set of objects whose uncertainty intervals at time $t$ intersect $P$.

## 5.4 Uncertainty and Communication Tradeoffs in Moving Objects Databases

Although the database location deviates from the actual location of a moving object, more frequent updates can

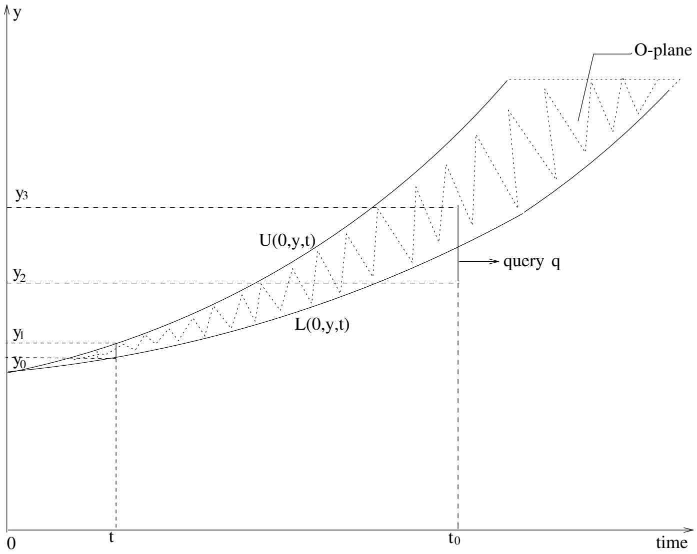


**Figure 3. Object $o$ is traveling along the $y$ axis. ($y_0, y_1$) is the uncertainty interval at time t. The query $q$ (represented by the solid line interval) is: retrieve the objects which at time $t_0$ are at $x = 0$ between $y_2$ and $y_3$.**

reduce the deviation. Clearly there is a tradeoff between communication and imprecision in the sense that the higher the number of updates the lower the imprecision, and vice versa. In the model that we presented in subsection 5.1, the imprecision is captured by the bound on the deviation. The main issue addressed in this subsection is how to determine the bound, denoted $L.uncertainty$.

We take a cost based approach to solve this problem.

### 5.4.1 The Information Cost of a Trip

The information cost of a trip has the following three components:

- **Deviation Cost**

  The deviation has a cost (or penalty) because it can result in incorrect decision making. Observe first that the cost of the deviation depends both, on the size of the deviation and on the length of time for which it persists. It depends on the size of the deviation since decision-making is clearly affected by it. To see that it depends on the length of time for which the deviation persists, suppose that there is one query that retrieves the location of a moving object $m$ per time unit. Then, if the deviation persists for two time units its cost will be twice the cost of the deviation that persists for a single time unit; the reason is that two queries (instead of one) will pay the deviation penalty. Formally, for a moving object $m$ the cost of the deviation between two time points $t_1$ and $t_2$ is given by the *deviation cost function*, denoted $COST_d(t_1, t_2)$; it is a function of two variables that maps the deviation between the time points $t_1$ and $t_2$ into a nonnegative number. In this paper we take the penalty for each unit of deviation during a unit of time to be one (1). Then, the cost of the deviation between two time points $t_1$ and $t_2$ is:

$$COST_d(t_1, t_2) = \int_{t_1}^{t_2} d(t)dt \qquad (1)$$

where $d(t)$ is the deviation as a function of time.

- **Update Cost**

  The *update cost*, denoted $C_1$, is a nonnegative number representing the cost of a location-update message sent from the moving object to the database. The update cost may differ from one moving object to another, and it may vary even for a single moving object during a trip, due for example, to changing availability of bandwidth. The update cost must be given in the same units as the deviation cost. In particular, if the update cost is $C_1$ it means the ratio between the update cost and the cost of a unit of deviation per unit of time (which is one) is $C_1$. It also means that the moving object (or the system) is willing to use $1/C_1$ messages in order to reduce the deviation by one during one unit of time.

- **Uncertainty Cost**

  The uncertainty has a cost (or penalty) because a higher uncertainty conveys less information when answering a query. Observe that, as for the deviation, the cost of the uncertainty depends both, on the size of the uncertainty and on the length of time for which it persists. Formally, for a moving object $m$ the cost of the uncertainty between two time points $t_1$ and $t_2$ is given by the *uncertainty cost function*, denoted $COST_u(t_1, t_2)$; Define the *uncertainty unit cost* to be the penalty for each unit of uncertainty during a unit of time, and denote it by $C_2$. Thus $C_2$ is the ratio between the cost of a unit of uncertainty and the cost of a unit of deviation. Then, the cost of the uncertainty of a moving object $m$ between two time points $t_1$ and $t_2$ is:

  $$COST_u(t_1, t_2) = \int_{t_1}^{t_2} C_2 u(t)dt \qquad (2)$$

  where $u(t)$ is the value of the $L.uncertainty$ subattribute of $m$ as a function of time.

Now we are ready to define the information cost of a trip taken by a moving object $m$. Let $t_1$ and $t_2$ be the timestamps of two consecutive location update messages. Then the *information cost* in the half open interval $[t_1, t_2)$ is:

$$COST_I[t_1, t_2) = C_1 + COST_d[t_1, t_2) + COST_u[t_1, t_2) \qquad (3)$$

and the *total information cost* of the trip is:

$$COST_I = COST_d[0, t_1) + COST_u[0, t_1) + \sum_{i=1}^{k} COST[t_i, t_{i+1}) \qquad (4)$$

where $t_1, t_2, ..., t_k$ are the time points of the update messages sent by $m$, 0 is the time point when the trip started, and $t_{k+1}$ is the time point when the trip ended.

### 5.4.2 Descriptions of Update Policies

The objective of the location update policies that we discuss in this paper is to set the deviation bound (or threshold) of a moving object, namely its $L.uncertainty$ subattribute, such that the total information cost is minimized. Due to space limitations we only outline the main ideas in our update policies. For a complete discussion see [33].

- **The Speed Dead-reckoning (sdr) Policy.**

  At the beginning of the trip the moving object $m$ sends to the DBMS an uncertainty value that is selected in an ad hoc fashion, it is stored in $L.uncertainty$, and it remains fixed for the duration of the trip. The object $m$ updates the database whenever the deviation exceeds $L.uncertainty$; the update simply includes the current location and current speed. [2]

- **The Adaptive Dead Reckoning (adr) Policy.**

  When using the adr policy, a moving object provides with each update a new uncertainty value $th$ that is computed using a cost based approach. $th$ minimizes the total information cost, i.e. the sum of the update cost, the deviation cost, and the uncertainty cost. At location update time, in order to compute the new uncertainty value, $m$ predicts the behavior of the deviation. The uncertainty values differ from update to update because the predicted behavior of the deviation is different. Our analysis indicates that the optimum uncertainty value is $\sqrt{\frac{2aC_1}{2C_2+1}}$, where $a$ is the approximated slope of the deviation, $C_1$ is the update cost, and $C_2$ is the uncertainty unit cost.

- **The Disconnection Detection Dead Reckoning (dtdr) Policy.**

  A problem in our model is that the moving object may be disconnected or otherwise unable to generate location updates. In other words, although the DBMS "thinks" that updates are not generated because the deviation does not exceed the uncertainty value, the actual reason is that the moving object is disconnected. To cope with this problem we introduce a third policy, "disconnection detecting dead-reckoning (dtdr)". The policy uses a novel technique that decreases the uncertainty value for the purpose of disconnection detection. Thus, in dtdr the uncertainty value continuously decreases as time since the last location update passes. It has a value $K$ during the first time unit after the update, it has value $K/2$ during the second time unit after

[2] Sdr can also use another speed, for example, the average speed since the last update, or the average speed since the beginning of the trip, or a speed that is predicted based on knowledge of the terrain. This comment holds for the other policies discussed in this section.

the update, it has value $K/3$ during the third time unit, etc. Thus, if the object is connected, it is increasingly likely that it will generate an update. Conversely, if the moving object does not generate an update, as time since the last update passes it is increasingly likely that the moving object is disconnected. The dtdr policy computes the $K$ that minimizes the total information cost, i.e. the sum of the update cost, the deviation cost, and the uncertainty cost.

To contrast the three policies, observe that for sdr the uncertainty values are fixed for all location updates. For adr the uncertainty values are fixed between each pair of consecutive updates, but they may change from pair to pair. For dtdr each uncertainty value decreases as the period of time between a pair of consecutive updates increases.

#### 5.4.3 Simulation Results

We conducted numerous simulations to compare the information cost of location update policies. The parameters of the simulation are the following: the update-unit cost, namely the cost of a location-update message, the uncertainty-unit cost, the deviation-unit cost, and a speed curve, namely a function that for a period of time gives the speed of the moving object at any point in time. We built a simulation testbed which enables us to compare the policies in terms of number of messages, deviation, and uncertainty.

We compared by simulating the policies adr, dtdr and sdr. The comparison is done by quantifying the total information cost of each policy for a large number of combinations of the parameters. Our simulations indicate that adr is superior to sdr in the sense that it has a lower or equal information cost for every value of the update-unit cost, uncertainty-unit cost, and deviation-unit cost. Adr is superior to dtdr in the same sense; the difference between the costs of the two policies quantifies the cost of disconnnection detection. For some parameters combinations the information cost of sdr is six times as high as that of adr.

## 6 PROTOTYPE DESIGN

We implemented a prototype which packages all the capabilities we have discussed for MOD applications. The prototype implements the capabilities that should be added to a central DBMS to support MOD applications, the software on the moving object, and a simulation system to evaluate the cost of location update policies. As shown in Figure 4, this prototype has four subsystems, i.e. Query Processing Subsystem(QPS), Moving Object Subsystem(MOS), Policy Simulation Subsystem(PSS), and Update Policies Subsystem(UPS). We built these subsystems in a modular fashion so that each can work independently, and at the same time they can be combined in various ways into different functional packages.

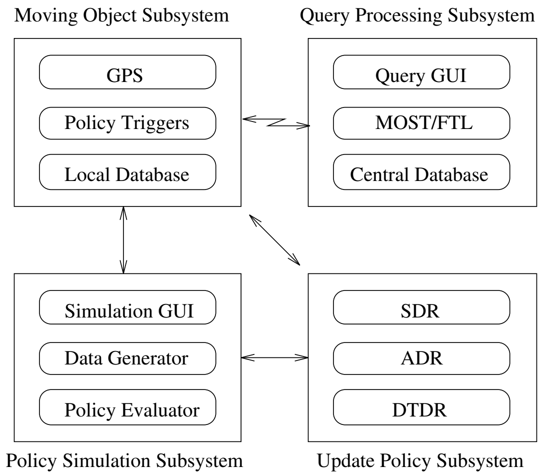


**Figure 4. The architecture of the DOMINO prototype**

- **Query Processing Subsystem(QPS)**

  We implemented the MOST data model and the FTL language on top of Sybase and MS Windows. The Query GUI is used to enter FTL queries and triggers, and view the query results. Any query or trigger is first examined (and possibly modified) by the MOST/FTL system, and so is the answer of the DBMS before it is returned to the user. For a detailed discussion on modifications to queries and answers of the underlying DBMS see [14].

  In the current prototype, there is a Central Database which is updated with the location of all the moving objects. In the future we intend to consider another case in which the location database is distributed and partially replicated among the moving objects.

- **Moving Object Subsystem(MOS)**

  The MO subsystem prototypes the local system on a moving object. It implements update policies using database triggers. The current location of the object is kept in a local database and it is updated by a GPS at a fixed rate (e.g. 1 sec.). The Local Database is managed by a DBMS which supports triggers (currently we use the Informix Universal Server). A trigger fires and updates the Central Database when the deviation bound is reached. The trigger also invokes the UP subsystem to compute a new deviation bound.

- **Update Policies Subsystem(UPS)**

  This subsystem implements update policies and their evaluation algorithms. It enables the definition of new policies without affecting other parts of the overall system.

- **Policy Simulation Subsystem(PSS)**

  PSS is intended to evaluate different update policies in terms of the total cost of information, the number of updates, the uncertainty, and the deviation. It uses as input a speed curve, namely a curve that gives the speed as a function of time. The Simulation GUI allows users to enter a speed curve by plotting it on the screen.

For each speed curve, update policy, update cost $C_1$, and uncertainty unit cost $C_2$ the PS subsystem executes a simulation run. The run computes the information cost [3] (a single number) of the policy on the curve. Then, for each policy, PSS averages the information cost over all the speed curves, and plots this average as a function of the update cost $C_1$.

Each simulation run is executed as follows. A speed-curve is a sequence $S$ of actual speeds, one for each time unit. Using $S$, PSS simulates the moving object's computer working with a particular update policy. This is done as follows. For each time unit there is an uncertainty value $th$, as well as a database speed and an actual speed. The actual speed is given by the speed curve. The deviation at a particular point in time $t$ is the difference between the integral of the actual-speed as a function of time, and the integral of the database-speed (the integrals are taken from the time of the last update until $t$). Denote by $T$ the sequence of deviations, one at each time unit. Denote by $Q$ the sequence of uncertainty values (or thresholds), one at each time unit. If the deviation at time $t$ reaches the uncertainty value, then PSS generates an update record consisting of: the current time, the current location, the current speed, the next uncertainty (for adr and dtdr it is computed as explained in the previous section); the deviation at time $t$ becomes zero. Denote by $U$ the sequence of update records. Using $T$, PSS computes the total cost of deviation, denoted $c_1$, and using $U$ we compute the total cost of updates, $c_2$. Using $Q$ we computes the total cost of uncertainty, $c_3$. The information cost of the policy on the speed curve is $c_1 + c_2 + c_3$.

At each location update, the PS subsystem invokes the UP subsystem to compute the new deviation bound. The PS subsystem keeps the speed files and the simulation results, and provides the tools to aggregate the simulation results. In addition, the PS subsystem allows a user to modify parameters during a simulation run.

In the rest of this section, we discuss how the four subsystems can be combined into different functional packages. We are working on making these packages available over the WWW.

- **Policy Simulation + Update Policies**

  This combination provides an update policy simulation testbed, from which we got the simulation results discussed in subsection 5.4.3.

[3] Remember, the *information cost* of an update policy on a given speed-curve is computed by using equation 3 for every time interval between two consecutive update points.

- **Moving Object + Update Policies**

  This combination prototypes a moving object in real life. It is used to generate location updates from a moving object.

- **Moving Object + Update Policies + Query Processing**

  This combination is used to query a set of real moving objects. Location data is generated by each moving object. The MO subsystem updates the Central Database when deviation bound is reached, and invokes the UP subsystem to compute a new bound. The QP subsystem is used to query the central database.

- **Comprehensive Simulation**

  We intend to integrate the current simulation testbed with a GIS (Geographic Information System) that manages a geographic region, and with a model of wireless bandwidth allocation in the region. This will provide a comprehensive simulation driven prototype that can be used to evaluate the capacity of a MOD system, e.g. answer the query: how many mobile units can be supported for a given level of location accuracy and a given percentage of available bandwidth for location updates; or, what bandwidth is necessary to support a 90% accuracy for 10,000 objects. It can also be used for evaluating the performance of queries and triggers in a centralized and distributed environment.

# 7 RELEVANT WORK

To the best of our knowledge, this is the first project in which the critical issues in moving objects databases are systematically addressed. Furthermore, the issues do not seem to fit neatly into an established field of research. Nevertheless, several research areas are relevant to the project.

One area of research that is relevant to the model and language presented in this paper is temporal databases [8, 32, 29]. The main difference between our approach and the temporal database works is that, by and large, those works assume that the database varies at discrete points in time; and between updates the values of database attributes are constant ( [29] uses interpolation functions to some extent). In contrast, here we assume that dynamic attributes change continuously, and consequently the temporal data model is different than the data model presented in this paper. It is not clear if and how temporal extensions to deal with incomplete information (see [7, 15]) are applicable to our context. However, temporal languages other than FTL can be used to query MOST databases. Nevertheless, when using any other language, the query processing algorithm will have to be modified to handle dynamic attributes.

Another relevant area is spatial databases (see [28, 18, 11, 9, 10, 12, 17]). Work in this area can be used for defining and processing the spatial operators discussed in section 3.

Our work is also relevant to uncertainty in databases (see [1, 25] for surveys). However, as far as we know this area has so far addressed complementary issues to the ones in this paper. Our current work addresses the question: what uncertainty to initially associate with the location of each moving object. In contrast, existing works are concerned with management and reasoning with uncertainty, after such uncertainty is introduced in the database. Thus these works become important for query processing that goes beyond "may" and "must" semantics.

Another body of relevant work is constraint databases (see [22] for a survey and [16, 4] for some notable systems). Constraint databases have been separately applied to the temporal domain, and to the spatial domain. Constraint databases can be used as a framework in which to implement dynamic attributes. A constraint will have to be adapted to provide a single value at each point in time, with the values changing over time. It will also have to be adapted for movement on routes and for uncertainty (see [23]).

An interesting relevant approach to modeling the location of moving objects is taken in [13]. In contrast to our approach which addresses tracking of moving objects, i.e. querying the current and future locations, the [13] approach pertains to past histories of moving objects. Thus its modeling viewpoint is different, and it does not address indexing, imprecision, and uncertainty.

Another relevant research concerns location management for mobile users in the cellular architecture (see [30, 19, 3, 21, 20, 2, 24]). When calling a mobile user, the Personal Communication Service (PCS) infrastructure must locate the cell in which the user is currently located. The above works address the problem of allocating and distributing the location database (i.e. the database that gives the current cell of each mobile user) such that the lookup time and update overhead are minimized. The location of a user is given at the granularity of a whole cell, which is sufficient for the purpose of calling a mobile user. For a given network, the cells are of a fixed size. In wide-area networks, the diameter of a cell ranges from a couple of miles in the terrestrial architecture, to thousands of miles in satellite architectures. In other words, for PCS communication the location uncertainty is always fixed, and is given by the size of the cell. However, what happens when mobile users are not covered by a cellular architecture (e.g. in the Desert-Storm battlefield) and/or when the location-uncertainty of the cellular network is too large (e.g., for picking up a customer, knowing the location of a taxi-cab withing 10 miles may not be satisfactory). For these cases, the existing work on location management is not satisfactory.

# 8 FUTURE WORK

Much remains to be done in order to make moving objects a commercial reality. We intend to extend the present work in the following directions:

- In some cases MOD applications may not be interested in the *location* of moving objects at any point in time, but in their arrival at the destination by a particular deadline. Assume that the database arrival information is given by "The object is expected to (or must) arrive within 10 minutes of 5pm". How do our current results apply to this case? We believe that most of the them carry over. However, we need make adjustments in order to handle the increasing criticality and better estimation capability as the 5pm deadline approaches.

- Extend the present work to handle uncertainty for moving objects that do not report their location; instead their location is sensed by possibly unreliable means. This is the case, for example, for enemy forces in a battlefield.

- Extend our query processing and update policies to an environment without the central database, i.e. where database is distributed among the moving objects. Efficient location replication strategies need to be developed to for distributed model.

- Study the implications of the network QoS(Quality of Service) on update policies and query processing. The relevant QoS parameters are available bandwidth, message loss rate, average message delay, etc.

- We intend to integrate the current simulation testbed with a GIS (Geographic Information System) that manages a geographic region, and with a model of wireless bandwidth allocation in the region. This will provide a comprehensive simulation driven prototype that can be used to evaluate the capacity of a MOD system, e.g. answer the query: how many mobile units can be supported for a given level of location accuracy and a given percentage of available bandwidth for location updates; or, what bandwidth is necessary to support a 90% accuracy for 10,000 objects. It can also be used for evaluating the performance of queries and triggers in a centralized and distributed environment.

We believe that as the world becomes a more dynamic place, as geographic distances are shrinking and remote locations of the globe become more accessible, and as new applications are being developed, moving objects databases will become increasingly important.

# References


[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] R. Alonso and H. F. Korth. Database system issues in nomadic computing. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, May 1993.

[3] B. R. Badrinath, T. Imielinski, and A. Virmani. Locating strategies for personal communication networks. *Workshop*

*on Networking for Personal Communications Applications, IEEE GLOBECOM*, Dec. 1992.

[4] A. Brodsky, V. E. Segal, J. Chen, , and R. A. Exarkhopoulo. The ccube constraint object-oriented database system. *manuscript*, 1997.

[5] S. Chamberlain. Automated information distribution in bandwidth-constrained environments. *MILCOM-94 conference*, 1994.

[6] S. Chamberlain. Model-based battle command: A paradigm whose time has come. *1995 Symposium on C2 Research & Technology, NDU*, June 1995.

[7] C. Dyreson and R. Snodgrass. Temporal deductive databases and infinite objects. *ACM Symposium on Principles of Database Systems*, March 1988.

[8] R. S. ed. Special issue on temporal databases. *Data Engineering*, Dec. 1988.

[9] M. J. Egenhofer. Interaction with geographic information system via spatial queries. *J. Visual Languages and Computing*, 1(4), 1990.

[10] M. J. Egenhofer. Extending sql for cartographic display. *Cartography and Geographic Information Systems*, 18(4), 1991.

[11] M. J. Egenhofer. Spatial sql: A query and presentation language. *IEEE Transaction on Knowledge and Data Engineering*, 6(1), 1994.

[12] M. J. Egenhofer and R. Franzosa. Towards a spatial query language: User interface considerations. *Proc. 14th Int. Conf. VLDB*, 1988.

[13] M. Erwig, R. H. Guting, M. Schneider, M., and Varzigiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *First Chorochronos Intensive Workshop on Spatio-Temporal Database Systems*, 1997.

[14] O. Etzion, S. Jajodia, , S. Sripada, and eds. *Temporal Databases: Research and Practice*. Springer Verlag, 1998.

[15] Y.-C. P. S. Gadia and S. Nair. Incomplete information in relation temporal databases. *Eighteenth VLDB*, Aug. 1992.

[16] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin. Dedale, a spatial constraint database. *manuscript*, 1997.

[17] O. Guenth and A. Buchmann. Research issues in spatial databases. *SIGMOD Rec.*, 19(4), 1990.

[18] R. Guting. An introduction to spatial database systems. *VLDB Journal*, 4 1994.

[19] J. S. M. Ho and I. F. Akyildiz. Local anchor scheme for reducing location tracking costs in pcn. *1st ACM International Conference on Mobile Computing and Networking(MOBICOM'95)*, Nov. 1995.

[20] T. Imielinski and H. Korth. *Mobile Computing*. Kluwer Academic Publishers, 1996.

[21] R. Jain, Y.-B. Lin, C. Lo, , and S. Mohan. A caching strategy to reduce network impacts of pcs. *IEEE Journal on Selected Areas in Communications*, 12, Oct. 1994.

[22] P. Kanellakis. Constraint programming and database languages. *ACM Symposium on Principles of Database Systems*, May 1995.

[23] M. Koubarakis. Linear constraint databases for indefinite spatiotemporal information. *First Chorochronos Intensive Workshop on Spatio-Temporal Database Systems*, 1997.

[24] Lazoff, B. Stephens, and Y. Yesha. Optimal location of broadcast sources in unreliable tree networks. *IEEE International Conference on Computers and Communications Networks*, 1996.

[25] A. Motro. Management of uncertainty in database systems. *In Modern Database Systems, Won Kim ed., Addison Wesley*, 1995.

[26] OmniTRACS. Communicating without limits. *http://www.qualcomm.com/ProdTech/Omni/prodtech/omnisys.html*.

[27] H. Samet. *The design and analysis of spatial data structures*. Addison Wesley, 1990.

[28] H. Samet and W. Aref. Spatial data models and query processing. *In Modern Database Systems, Won Kim ed., Addison Wesley*, 1995.

[29] A. Segev and A. Shoshani. Logical modeling of temporal data. *Proc. of the ACM-Sigmod International Conf. on Management of Data*, 1987.

[30] N. Shivakumar, J. Jannink, and J. Widom. Per-user profile replication in mobile environments: Algorithms, analysis, and simulation results. *to appear ACM/Baltzer Journal on Special Topics in Mobile Networks and Applications, special issue on Data Management*, 1997.

[31] P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. *Proceedings of the Thirteenth International Conference on Data Engineering(ICDE13)*, Apr. 1997.

[32] R. Snodgrass and I. Ahn. The temporal databases. *IEEE Computer*, Sept. 1986.

[33] O. Wolfson, L. Jiang, A. P. Sistla, S. Chamberlain, and M. Deng. Updating and probabilistic querying of motion databases. *submitted for publication*, 1998.