Boosting Location-Based Services with a Moving Object Database Engine

Nikos Pelekis
Dept of Informatics
Univ. of Piraeus, Hellas
npelekis@unipi.gr

Yannis Theodoridis
Dept of Informatics
Univ. of Piraeus, Hellas
ytheod@unipi.gr

ABSTRACT

Composition of temporal and spatial properties of real world objects in a unified data framework results into Moving Object Databases (MOD). MODs are able to process, manage and analyze discretely or continuously changing spatio-temporal data. This paper presents HERMES Moving Data Cartridge, which provides MOD functionality to OpenGIS-compatible state-of-the-art Object-Relational DBMS. HERMES is designed to be used as a pure temporal or a pure spatial system, however, its main application is to support modeling and querying of moving objects. A relevant collection of abstract data types (ADT) and their corresponding operations are defined, developed and provided as a data cartridge extending SQL-like query languages with MOD semantics. The usefuleness of the resulting query language is demonstrated by developing an application on top of this framework, which builds and visualizes the results of a palette of spatio-temporal queries that have been proposed in the literature as an advanced Location-Based Services (LBS) benchmarking framework for the evaluation of MOD engines.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems

General Terms: Design

Keywords: HERMES, Moving Object Databases, Location-Based Services, Data Cartridge, Benchmark Queries.

1. INTRODUCTION

Spatial database research has focused on supporting the modeling and querying of geometries stored in a database. On the other hand, temporal databases have focused on extending the knowledge kept about the current state of the real world to include the past, in the two senses of "the past of the real world" (valid time) and "the past states of the database" (transaction time). About a decade efforts attempt to achieve an appropriate kind of interaction between both sub-areas of database research. Spatio-temporal databases are the outcome of the aggregation of time and space into a single framework [15], [1], [16], [10].

MobiDE'06, June 25, 2006, Chicago, Illinois, USA.

As delineated in the review papers just cited, a serious weakness of existing approaches is that each of them deals with few common characteristics found across a number of specific applications. Thus, the applicability of each approach to different cases fails on spatio-temporal behaviors not anticipated by the application used for the initial model development. The aim of this paper is to describe a robust framework capable of aiding a spatio-temporal database developer in modeling, constructing and querying a database with objects that change location, shape and size, either discretely or continuously in time. Objects that change location continuously are much more difficult to accommodate in a database in contrast to discretely changing objects. Supporting both types of spatio-temporal objects (the so-called *moving objects*) is one of the challenges adopted by this paper.

In particular, we present an integrated and comprehensive design of moving object data types in the form of a data cartridge, called HERMES Moving Data Cartridge (*HERMES-MDC*). HERMES-MDC is the core component of the object-relational part of the HERMES system architecture [14]. HERMES provides the functionality to construct a set of moving, expanding and/or shrinking geometries, modeled as sequences of simple continuous functions that obtain hypostasis when projected to the spatial domain at a specific instant associated with of time. Each one of these geometries is casted with a set of methods that facilitate the cartridge user to query and analyze spatio-temporal data. Embedding this functionality offered by HERMES-MDC in the data manipulation language of an OpenGIS-compatible state-of-the-art object-relational DBMS, one obtains a moving object query language that outperforms related work, in terms of flexibility, expressiveness and ease of use.

One could mention a series of applications of HERMES (from the name of the ancient Greek god of Commerce) at various levels in the context of mobile services. For example, HERMES can be used as a plug-in in telecom data warehouses that handle spatio-temporal content. This example refers to offline processing of such historical data. Besides, HERMES supports the data management of real-time mobile services, addressing the issues of emerging online applications. For instance, imagine a user (traveler, consumer, etc.) moving around a city with a high technology mobile terminal at hand (e.g. a smartphone or PDA equipped with a GPS receiver), receiving hints of information, commercial spots etc. Motivated from such kind of application scenarios, recent research has tried to model spatio-temporal databases using this concept of moving objects and integrate them into any extensible DBMS [3], [4], [6], [8]. On the other hand, commercial relational or object-relational database systems offer limited capability of handling this kind of non-traditional data (object trajectories, in time and space). HERMES is the partial

realization of the above discussed research vision in state-of-the-art Object-Relational DBMS.

In order to demonstrate the usefulness and applicability of the server-side extensions provided by HERMES we implement an LBS application on top of this MOD functionality. The general idea is to provide a flexible linkage for a non-expert user to pose a palette of MOD queries that have been proposed in the literature [19] as an advanced (LBS) benchmarking framework for the evaluation of MOD engines. To the best of our knowledge, this is the first work trying to boost LBS utilizing a MOD engine.

In the rest of the paper, we first present our data type model introduced by HERMES-MDC (Section 2), and then we propose an appropriate set of operations for the above types that extend SQL with MOD semantics (Section 3). For evaluation purposes, HERMES-MDC is applied to the previously discussed LBS case study (Section 4). Subsequently, the paper presents related work in the field in comparison with HERMES-MDC functionality (Section 5). Finally, the paper winds up and at the same time points out some interesting future research directions (Section 6).

2. HERMES MOVING DATA CARTRIDGE

In this section, we design a data type model for the endorsement of extending a query language with constructs that would enable the querying of MODs. We focus on capturing spatio-temporal processes that change continuously as this is the most challenging and also allow us to capture spatio-temporal phenomena that change in discrete steps as a special case of continuous change. The data types are classified into two main categories. The first category consists of off-the-shelf base, (static) spatial and temporal types and the second category introduces types that describe moving objects.

2.1 Base, Temporal and Spatial Types

Base types are the standard database types built into most DBMS, such as integer, real numbers etc. These types form a subset of the atomic literal types needed to define the temporal types. Temporal types are introduced by TAU Temporal Literal Library (*TAU-TLL*) in [13], which is the component of HERMES system responsible for providing HERMES-MDC with pure temporal object-relational functionality. Basically, this cartridge implements the *Time Model*, adopted by the *TAU Temporal Object Model* [9], and augments the four temporal literal data types found in *ODMG* object model [2] (namely, *Date*, *Time*, *Timestamp* and *Interval*) with three new temporal object data types (namely, *Timepoint*, *Period* and *Temporal Element*). TAU-TLL provides clear semantics about the time boundaries, time order, time reference, temporal granularities, and the supported calendar. On the other hand, static spatial types are supported by a data cartridge providing an integrated set of functions and procedures that enable spatial data to be stored, accessed, and analyzed quickly and efficiently, such as Oracle Spatial [12].

2.2 Moving Types

As discussed in [20], the data obtained from moving point objects is similar to a "string", arbitrary oriented in 3D space, where two dimensions correspond to 2D (x-, y-) plane and one dimension corresponds to time. Instead of a "string" and due to discretization, a MOD stores and manipulates a 3D polyline representing the trajectory of the object (i.e., a sequence of 3D line segments, where each segment represents the continuous development of the moving object during sampled locations).

This idea is extended by HERMES-MDC in a way that a moving point can be defined as a sequence of different types of simple functions. The general idea is to decompose the definition of each moving type into several definitions, one for each of the simple functions, and then compose these sub-definitions as a collection to define the moving type. Each one of the sub-definitions corresponds to a so-called *unit* moving type. In order to define a unit moving type, we need to associate a period of time with the description of a simple function that models the behavior of the moving type in that specific time period. Based on this approach, two real-world notions are directly mapped to our model as object types, namely time *period* and *function*. The first concept (called *Period<SEC>* in TAU-TLL terminology [13]) implies a closed-open time interval (i.e. [*b, e*), where *b* is the beginning and *e* is the ending point of the period) with granularity at the second level (other granularities are also supported i.e. minute, hour etc). The second concept is an object type, named *Unit_Function*, defined as a triplet of (x, y) coordinates together with some additional motion parameters. The first two coordinates represent the initial (x_i, y_i) and ending (x_e, y_e) coordinates of the sub-motion defined, while the third coordinate (x_c, y_c) corresponds to the centre of a circle upon which the object is moving. Whether we have constant, linear or arc motion between (x_i, y_i) and (x_e, y_e) is implied by a *flag* indicating the type of the simple function. Since we require that HERMES manages not only historical data, but also online and dynamic applications, we further let a *Unit_Function* to model the case where a user currently (i.e., at an initial timepoint) is located at (x_i, y_i) and moves with initial velocity v and acceleration a on a linear or circular arc route. Figure 1 depicts a point moving with different kind of functions along subsequent temporal periods.

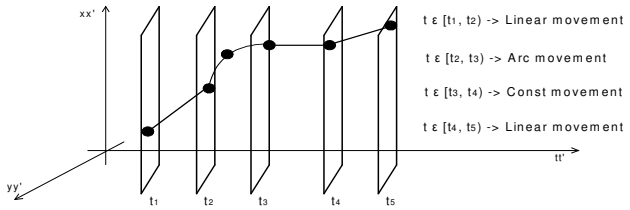

Figure 1 Moving Point with various types of movement

In the case of arc motions, following the categorization of realistic arc motions initially discussed in [23], we classify them according to the quadrant the motion takes place and motion heading (clockwise or counterclockwise). Figure 2 illustrates one of the possible eight cases (e.g. quadrant I - clockwise direction).

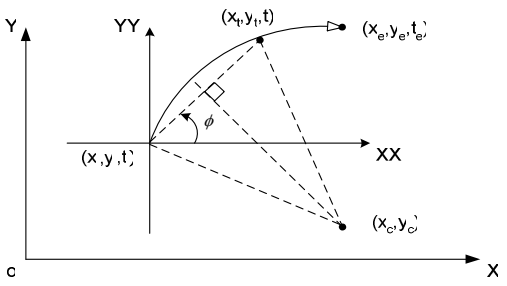

Figure 2 Motion on a circular arc

For constant and linear motions, the interpolation of a moving point's location in an intermediate timepoint t is straightforward. For arc motions, there is need of some trigonometric calculations. For the case of Figure 2 the necessary operations are illustrated in Eq. 1. Following a similar process, we develop all kinds of arc functions in each quadrant and direction.

$$ARC\_1(t) \Rightarrow (x_t, y_t) = (x_t + L_t \times Cos\phi, y_t + L_t \times Sin\phi)$$
$$L_t = 2 \times R \times Sin(\frac{S_t}{2 \times R})$$
$$S_t = v \times t + \frac{1}{2} \times a \times t^2 \qquad \textbf{Eq. 1}$$
$$\phi = \frac{\pi}{2} + Sin^{-1}(\frac{y_c - y_i}{R}) - \frac{S_t}{2 \times R}$$
$$v \geq 0, t \in [t_i, t_e), \phi \in (-\infty, +\infty)$$

Consequently, *Unit_Function* is defined as follows:

Definition 1: *Unit_Function* =
$_d\langle x_i$*:double,* y_i*:double,* x_e*:double,* y_e*:double,* x_c*:double,* y_c*:double, v:double, a:double, flag:TypeOfFunction*$\rangle$, where
Π *TypeOfFunction T={ CONST, PLNML_1, ARC_<1..8> }*

Combining *Unit_Function* and *Period<SECOND>* object types together, the most primitive and simplest unit object type is defined, namely *Unit_Moving_Point*. This is a fundamental type since all the successor unit types are defined based upon it. As such,

Definition 2: *Unit_Moving_Point* =
$_d\langle$*p: Period*$\langle$*SEC*$\rangle$*, m: Unit_Function*$\rangle$

Following this, we define two unit moving types directly based on *Unit_Moving_Point*, namely *Unit_Moving_Circle* and *Unit_ Moving_Rectangle*. As it is easily inferred, these two object types model circular and rectangular geometry constructs that change their position and/or extent over time. *Unit_Moving_Circle* consists of three *Unit_Moving_Point* objects, representing the three points (*f*, *s*, *t*) needed to define a valid circle. In the same way, *Unit_Moving_Rectangle* is composed of two *Unit_Moving_Point* objects, modeling the lower-left (*ll*) and upper-right (*ur*) point needed to define a valid rectangle. An intuitive constraint is that time periods during which these unit points are moving must be equal. More formally,

Definition 3: *Unit_Moving_Circle* =
$_d$ *{*$\langle$ *f: Unit_Moving_Point, s: Unit_Moving_Point, t: Unit_Moving_Point* $\rangle$ | *equal (f.p, s.p, t.p) }*

Definition 4: *Unit_Moving_Rectangle* =
$_d$ *{*$\langle$ *ll: Unit_Moving_Point, ur: Unit_Moving_Point* $\rangle$ | *equal (ll.p, ur.p) }*

For modeling object types such as *Unit_Moving_Polygon* and *Unit_Moving_LineString* there is need for an intermediate object type called *Unit_Moving_Segment* object, which models a simple line or arc segment that changes its shape and size according to its starting and ending points. As such, *Unit_Moving_Segment* is formed by three *Unit_Moving_ Point* objects and a flag indicating the kind of interpolation between the starting and ending point of the *LineString* geometry. The simplest part of a *LineString* geometry can be either a linear or an arc segment. In other words, this flag exemplifies the usage of the other attributes of *Unit_Moving_Segment*. This is clarified in Figure 3 where a moving segment is mapped to a line segment at two different time instants t_1 and t_2. During the time period between t_1 and t_2, the starting unit moving point mp_1 follows a simple linear trajectory, while the ending unit moving point mp_2 follows an arc trajectory.

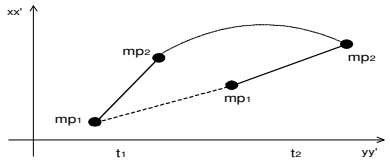

Figure 3 Linear Unit Moving Segment

Definition 5: *Unit_Moving_Segment* =
$_d$ *{*$\langle$ *b: Unit_Moving_Point, e: Unit_Moving_Point, m: Unit_Moving_Point, kind:TypeOfSegment* $\rangle$ | *(kind = SEG* $\Rightarrow$ *equal (b.p, e.p))* $\wedge$ *(kind = ARC* $\Rightarrow$ *equal (b.p, e.p, m.p)) },*
where Π *TypeOfSegment T = { SEG, ARC }*

Consequently, a *Unit_Moving_LineString* is defined as a set of *Unit_Moving_Segment* objects, while a *Unit_Moving_Polygon* is a collection of *Unit_Moving_Segment* objects, with an additional flag indicating if this set of moving segments forms the exterior ring of a polygon or an interior (hole) ring, which at any time does not cross or touch the exterior boundary. In terms of set theory,

Definition 6: *Unit_Moving_Linestring* =
$_d$ *{ l: set*$\langle$*Unit_Moving_Segment*$\rangle$ |
$\forall i, j \in ulong: i \neq j \Rightarrow equal (l_i.b.p, l_j.e.p) }$

Definition 7: *Unit_Moving_Polygon* =
$_d$ *{*$\langle$ *l: set*$\langle$*Unit_Moving_ Segment*$\rangle$*, hole:boolean* $\rangle$ |
$\forall i, j \in ulong: i \neq j \Rightarrow equal (l_i.b.p, l_j.e.p) }$

Having defined the fundamental unit moving types, we now introduce the moving types that play the dominant role in HERMES data type system. The process that is followed to define the moving types is to introduce a moving type as a collection of the corresponding unit moving type. Having this in mind, we construct a *Moving_Point* object type as a collection of *Unit_Moving_Point* objects, whose periods must be sequential and should not overlap. The projection of a *Moving_Point* to the spatial dimension should result to a valid point geometry. In other words,

Definition 8: *Moving_Point* =
$_d$ *{ tab: set*$\langle$*Unit_Moving_Point*$\rangle$ | $\forall i, j \in ulong, 1 \leq i, j \leq$ | *set*$\langle$*Unit_Moving_Point*$\rangle$ |*:* $j = i+1 \Rightarrow before(tab_i.p, tab_j.p) \wedge \neg overlaps(tab_i.p, tab_j.p) \wedge \forall t \in double: inside(t, tab_i.p) \Rightarrow at\_instant(t) \in Geometry(point)\}$

Similarly to the *Moving_Point*, other moving types are constructed as collections of their *unit* counterparts. Due to space limitations, we only present the definition of a *Moving_ Linestring*.

Definition 9: *Moving_LineString* =
$_d$ *{ line: set*$\langle$*Unit_Moving_ LineString*$\rangle$ | $\forall i, j \in ulong, 1 \leq i, j \leq$ | *set*$\langle$*Unit_Moving_ LineString*$\rangle$*:* $j = i+1 \Rightarrow before(line_i.l_1.b.p, line_j.l_1.e.p) \wedge \neg overlaps(line_i.l_1.b.p, line_j.l_1.e.p) \wedge \forall t \in double: inside(t, line_i.l_1.b.p) \Rightarrow at\_instant(line,t) \in Geometry(linestring) }$

The definition of *Moving_Polygon* is very close to that of *Moving_LineString*. Actually, the differences between these two

moving types arise from the different utilization of their collections of moving segments by the object methods. For example, an operation that maps a *Moving_LineString* to a *LineString* geometry checks for inequality on the starting and ending points of the line and this is a prerequisite for constructing the geometry. On the contrary, the corresponding method for a moving polygon checks for the opposite, in order to be able to construct a valid polygon. Another discrepancy of *Moving_Polygon*, in contrast to all the other moving types, is that in case it includes interior moving holes, then several *Unit_Moving_Polygon* objects need to be accessed in order to transform it to its corresponding OpenGIS-compatible spatial geometry at a specific instant.

Similarly, in order to model homogeneous collections of moving types, *multi*-moving types are defined as collections of the corresponding moving types. Consequently, the proposed spatio-temporal model is augmented by the following object types: *Multi_Moving_Point*, *Multi_Moving_Circle*, *Multi_Moving_Rectangle*, *Multi_Moving_LineString* and *Multi_Moving_Polygon*. An interesting issue here is that the previously mentioned multi-moving types do not carry their own methods interface. The functionality for these types can be invoked by the methods of another object type, called *Moving_Collection*, standing as the supertype and aggregating the interfaces, the object methods and the spatio-temporal semantics of all the multi moving types. Furthermore, *Moving_Collection* is able to represent heterogeneous collections of moving types. The methods of *Moving_Collection* treat all the multi moving types uniformly and they do not have the knowledge whether they are dealing with a homogeneous or heterogeneous collection. The formal definition of *Moving_Collection* is omitted as a trivial one.

We also introduce an object that encapsulates all semantics and functionality offered by all moving types. The so-called *Moving_Object* object type is the conjunction of all the other moving object types, which implies that this object can completely substitute any other moving type. Furthermore, *Moving_Object* models any moving type that can be the result of an operation between moving objects. For example, the intersection of a *Moving_Point* with a polygon geometry forms a second *Moving_Point* that is the restriction of the first *Moving_Point* inside the polygon. This result can be modeled as a *Moving_Object*. If the result of an operation is not a moving geometry then *Moving_Object* plays the role of a degenerated moving type. For example, let us assume an operation that requests the perimeter of *Moving_Polygon*; obviously, the result of this method is a time-varying real number (*Moving_Real*). Such collapsed moving types like moving real, string, and boolean are also modeled using the *Moving_Object* object type. *Moving_Object* is not intended to be directly used or constructed by HERMES user. On the contrary, it is intended to be the result type of operations of the other moving types (i.e., system generated). For a detailed description of *Moving_Object* type the interested reader is referred to [13]. Figure 4 illustrates the UML class diagram of the above defined moving object data types [14].

3. OPERATIONS ON MOVING OBJECTS

The design of the operations of the object types introduced by HERMES-MDC adheres to three principles: a) design operations as generic as possible; b) achieve consistency between operations on pure spatial, pure temporal and spatio-temporal types; c) capture the interesting phenomena.

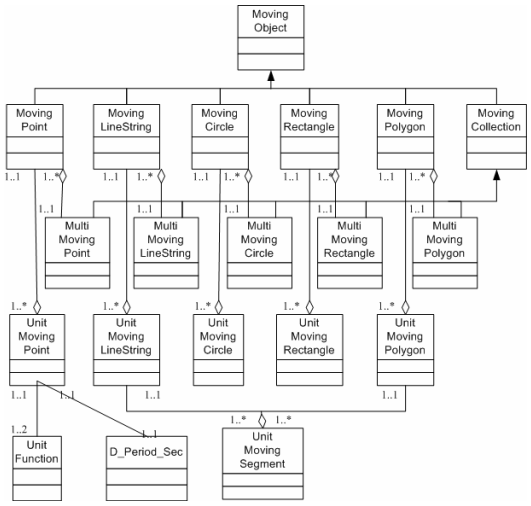

Figure 4 HERMES-MDC Class Diagram

For the first principle we focus on properties shared by many types. In order to achieve consistency of operations on spatial, temporal and moving types, we first study operations proposed in the literature for pure spatial types and we select those operations that we would like to associate with temporal semantics. In a second step, we use the functionality of the temporal types introduced by the temporal component of HERMES (TAU-TLL) and we systematically extend the operations defined in the first step to the temporal variants of the respective types. The third step takes the previous time-dependant operations as its outset and removes their time dimension thus not returning pure spatial, temporal or standard data types, but other moving types (e.g. the length of a moving linestring independently of a specific time point). Finally, to achieve closure of "interesting phenomena", our development is driven by state-of-the-art emerging applications such as LBS.

The following sections describe the functionality with which SQL-like query languages are enhanced by the use of HERMES functionality. The presentation of the operations hides the technical details that would disorient us from expressing the power of the resulted query language. As such, we abstractly describe the algorithms for only a motivating set of operations. Due to space limitations we focus our discussion on methods defined on *Moving_Point* type (as the most challenging and LBS relevant); however operations with similar semantics are also defined for the rest types.

3.1 Topological and Distance Predicates

HERMES-MDC provides object methods in the form of predicates to describe relationships between moving types. There are two sets of predicates supported by HERMES-MDC, namely *within_distance* and *relate*. Each set of predicates consists of eight operations, each of which models the relationship of the caller moving type with any time-varying (or not) geometry object. Each operation comes with two different overloaded signatures, modeling different semantics: the first signature is time-dependent while the second is independent to the time dimension. Below, the reader can find the pair of signatures of only one of the eight

operations. The time-dependent signature of the method is the one without the brackets, while the time-independent version of the operation can be obtained by substituting the return type of the operation with the type in the brackets { } and by removing the *Timepoint<SEC>* argument from the parameter list. This is a common notation in the remainder of the paper.

- *boolean {Moving_Object}* ***within_distance*** *(distance, Moving_Point, tolerance, Timepoint<SEC>)*

The time-dependent predicate determines whether two moving objects are within some specified Euclidean distance from each other at a user-defined time point. After mapping the moving objects to physical spatial geometries at the given instant, the function returns *true* for object pairs that are within the specified distance; returns *false* otherwise. The distance between two non-point objects (such as lines and polygons) is defined as the minimum distance between these two objects.

Many object methods in HERMES-MDC accept a tolerance parameter. If the distance between two points is less than or equal to the tolerance, the cartridge considers the two points to be a single point. Thus, tolerance is usually a reflection of how accurate or precise users perceive their spatio-temporal data to be. Also, the time-independent *within_distance* operation differs from the above predicate in that the return value is a *Moving_Object* that represents a time-varying boolean value. This implicitly defined "moving boolean" object models the sequence of the time intervals that the two related objects are within or not some specified Euclidean distance.

- *Varchar2 {Moving_Object}* ***relate*** *(mask, Moving_Polygon, tolerance, Timepoint<SEC>)*

This generic predicate examines two moving objects and determines their topological relationship. As previously, the *relate* predicate appears with two overloaded versions. The first evaluates the topological relationship upon a specific user-defined time point, while the second version returns a *Moving_Object* modeling a time-varying string, which describes the evolution in the topological relationship between the related objects. The user can specify the kind of any of the well-known topological relationships that he/she requires to check via the mask parameter.

3.2 Interaction with the Temporal and Spatial Domains

HERMES-MDC provides object methods for restricting and/or projecting moving types to the temporal and the spatial domain. Subsequently, we present the most important operations defined for *Moving_Point*.

- *Unit_Moving_Point* ***unit_type*** *(Timepoint<SEC>)*

The simple but very important task that this function performs is that it finds the unit moving object whose attribute time period "contains" the user-defined time point. In other words, it returns that unit-moving type where the time instant represented by the argument *Timepoint<SEC>* object is "inside" the time period that characterizes the unit-moving type.

- *Moving_Point* ***add_unit*** *(Unit_Moving_Point)*

This operation adds a new coming unit of movement as this is described by a *Unit_Moving_Point* object. Naturally, the method performs special consistency operations (e.g. the period of the argument must not overlap with the lifespan of the initial object) to assert the soundness of the constructed object.

- *Union_Output* ***at_instant*** *(Timepoint<SEC>)*

The *at_instant* operation is the operation that maps the moving types to meaningful OpenGIS-compatible spatial objects. The return type (*Union_Output*) is an object that represents the union of all the possible results of the projection of a *Moving_Object* at a user-defined time point. In other words, if *Moving_Object* represents a time-varying geometry then *Union_Output* is basically a *Geometry* object. If *Moving_Object* represents a "moving" real or string then *Union_Output* is a real number or a string, respectively. In the case of *Moving_Collection*, this operation invokes the *at_instant* operations of all the moving types of the multi moving objects and subsequently applies a special "union" operation upon the projected geometries by "concatenating" them in a collection object and returns the result of the "concatenation".

- *Moving_Point* ***at_period*** *(Period<SEC>)*

The *at_period* object method is an operation that restricts a moving object to the temporal domain. In other words, by using this function the user can delimit the time period that is meaningful to ask the projection of the moving object to the spatial domain. More specifically, the time period passed as argument to the method is compared with the *Period<SEC>* objects that characterize the unit moving objects. If the parameter period does not overlap with the compared period then the corresponding unit type is omitted. If it overlaps, then the time period that defines a unit-moving object becomes its "intersection" with the given period.

- *Temp_Element<SEC>* ***temp_element*** *()*

The *temp_element* operation gives HERMES-MDC user the capability to project the time periods that form the unit moving objects that compose a moving type on the time line and subsequently "concatenate" all these distinct time periods to construct a temporal element. Figure 5 depicts the result of the *temp_element* operation when applied to a *Moving_Point* object.

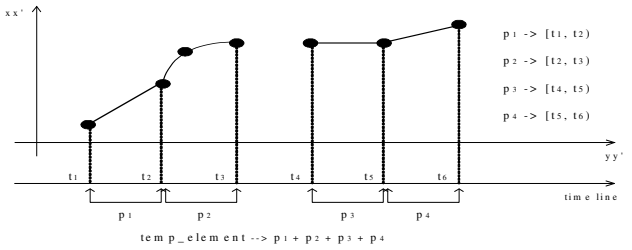

Figure 5 Projection of a Moving Point on the time axis

- *Moving_Point* ***at_temp_element*** *(Temp_Element<SEC>)*

Similarly to the *at_period* operation, the *at_temp_element* method restricts the moving object to the temporal domain, but the process of restricting the periods between which the moving object is valid is driven by a collection of *Period<SEC>* objects and not just one *Period<SEC>* object as in the previous case.

- *Moving_Point* ***at_linestring*** *(Geometry)*

Being aware that an object moves on a linestring geometry during a part of its route (e.g. a car moving along a street), we may wish to find the portion of the *Moving_Point* restricted by this 1-dimensional geometry. This is exactly the output of this method.

- *Union_Output* ***initial*** *()*

The *initial* object method is basically the *at_instant* operation invoked at the first instant of time that the moving object is valid, meaning the first second of the closed-open period that identifies the least recent unit moving object.

- *Union_Output* ***final*** *()*

Similarly to the *initial* object method, the *final* operation projects the moving object at the last valid instant of the time period that characterizes the most recent unit moving object.

- *Geometry* ***trajectory*** *()*

This operation reconstructs the trajectory traveled by a *Moving_Point*. More specifically, the operation projects the movement of a *Moving_Point* to the Cartesian plane by mapping the component *Unit_Moving_Point* objects to single linear or arc segments, while a process of merging these segments follows, to form the returned *LineString* geometry.

3.3 Distance and Direction Operations

The following methods assist the cartridge user to calculate the minimum distance or the directional relationship between moving objects.

- *number {Moving_Object}* ***distance*** *(Moving_Point, tolerance, Timepoint<Sec>)*

HERMES-MDC provides a distance measure that exists for all moving types, which either computes the distance between two instantiated moving objects (time-dependent version) or returns a time-varying real number that represents the minimum distance between these moving types at any time (time-independent version). The distance between two objects is the distance between the closest pair of points or segments of the two objects.

- *number {Moving_Object}* ***direction*** *(Moving_Point, Timepoint<Sec>)*

The *direction* function returns the angle of the line from the first to the second moving point (measured in degrees, $0° \le angle < 360°$), after these have been projected to the Cartesian plane at a specific time point. The time-independent version of the function returns a *Moving_Object* modeling a "moving real", which corresponds to the time-changing angle formed by the conceptual line segment that joins the two moving points and the *xx'* axis.

- *boolean* ***left*** *(Geometry, Timepoint<Sec>, from, to)*

The *left* operation returns true if the location of the point at the user defined timepoint is left from the argument geometry (i.e. the centroid in case of geometries with extent), which is the case when it falls inside the area formed by the argument angles *from* and *to*. Similarly, we define *right*, *front*, *behind* operations. Furthermore, we augment our operator set with a related set of methods that identify whether a moving point is located *west*, *east*, *north*, *south* of a geometry. These methods are differentiated from the previous as we do not care for the heading of the moving point.

3.4 Set Relationships

HERMES-MDC provides four object methods for describing set-relationships between moving types. Subsequently, we present only one (*intersection*) between a *Moving_Point* and a *Geometry*, while the rest of them (*union*, *difference*, *xor*) are omitted.

- *Geometry {Moving_Point}* ***intersection*** *(Geometry, tolerance, Timepoint<Sec>)*

Invoking *intersection* method for a *Moving_Point*, as one would expect, the result of this operation is the projection of itself on the spatial domain (point geometry) at time instants that intersects with other moving types or static geometries and null at time instants where it is not on the boundary or the interior of linestrings and polygons or it coincides with none of the points in a collection of them. The time-independent version returns the portion of the *Moving_Point* that intersects with the reference object. Figure 6 depicts the projection of a *Moving_Point* modeling its intersection with a polygon, at three different timepoints t_1, t_2, and t_3.

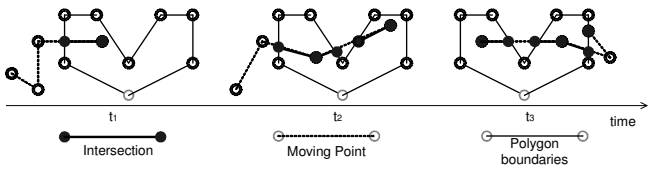

Figure 6 Demonstrating *intersection* operation

At timepoint t_1 it is obvious the result of such an operation is a linestring geometry. At timepoint t_2 this intersection has as result a multi-linestring geometry due to the development of *Moving_Point*, while at timepoint t_3 the resulted geometry is a heterogeneous collection of lines and points. This operation allows us to define methods returning the *entering/leaving* locations (as well as the respective timepoints) wherefrom a moving point passes when traversing spatial regions (Figure 7).

3.5 Rate of Change

An important property of any time-dependent value is its rate of change, i.e., its derivative. At least three properties of the *Moving_Point* permit the definition of derivative, namely the Euclidian distance, the direction and the vector difference (by viewing points as two-dimensional vectors). This leads to three different derivative operations, called *speed*, *turn* and *velocity*, respectively.

Due to space constraints, we omit signatures and descriptions of these three operations. The reader interested in these operations as well as in other operations as the area *traversed* by a moving polygon, building moving *buffer* of specific width around a path, constructing moving points from the *centroid* of moving areas, and finding the *num_of_components* of collections of moving types is referred to [13].

4. HERMES LBS TOOL

HERMES-MDC has been developed [14] as a system extension that provides MOD functionality to Oracle10g Object-Relational DBMS and, as such, the cartridge functionality extends PL/SQL

[7]. In order to demonstrate the usefulness and applicability of the server-side extensions provided by HERMES we implement a prototype application on top of this functionality which provides a graphical means to realize the majority of benchmark queries for LBS proposed in [19]. Especially, we develop an LBS application scenario for travelers entering the area of an airport, construct a spatial database modeling the ground plan of the airport, and input random trajectories of travelers moving around the area. The tool provides the ability to pose queries following the same classification as proposed in [19]. The idea is that a user selects one from a palette of such queries and according to his/her choice a wizard drives the user to parameterize his request. A query builder dynamically constructs the query using HERMES-MDC operations, sends the query to the server and visualizes the results using MapViewer [12].

Indicative supported queries include:

- *Queries on stationary reference objects*;
  - *point* (e.g. does this check-in serve my flight?),
  - *range* (e.g. are there any fellow travelers in the area in front of this check-in?),
  - *distance-based* (e.g. find the closest check-in),
  - *nearest-neighbor* (e.g. find the closest coffee shops to my current location) and
  - *topological queries* (e.g. find travelers crossed this gate during the past hour);
- *Queries on moving reference objects*;
  - *distance-based* (e.g. find travelers passed close to me this evening) and
  - *similarity-based queries* (e.g. find the three most similar trajectories to the one I have followed so far in the airport);
- *Join queries*;
  - *distance-join* (find the closest check-ins to travelers of this flight) and
  - *similarity-join queries* (find the two most similar pairs of travelers' trajectories);
- *Queries involving unary operators*, such as traveled distance or speed (e.g. find the average speed of travelers on Saturday nights).

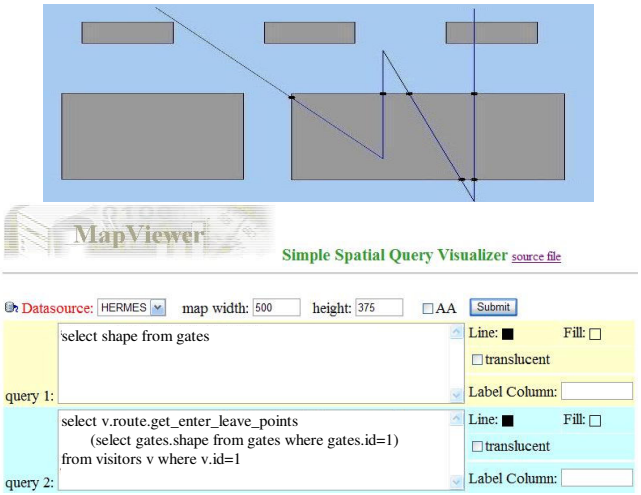

Figure 7 Visualization of enter/leave points in an area of interest

5. COMPARISON WITH RELATED WORK

Several research efforts have tried to model spatio-temporal databases using the concept of ADTs for moving objects. Such types for moving points and moving regions have been introduced by Güting and colleagues in [8], together with a set of operations on such entities. This model was the first attempt to deal with continuous motion, while in [6] the definition of the discrete representation of the above-discussed ADTs was presented. The next step in this development was the study of algorithms for the rather large set of operations defined in [8]. Whereas [6] just provides a brief look into this issue, in [11] the authors present a comprehensive, systematic study of algorithms for a subset of the operations introduced in [8]. This paper also proposed a blueprint for implementing such a *"moving objects"* extension package for suitable extensible database architectures.

Another model using moving objects is proposed by Wolfson and colleagues in [17], [22] and [21]. The authors propose the so-called Moving Objects Spatio-Temporal (MOST) data model for databases with *dynamic attributes*, i.e. attributes that change continuously as a function of time, without being explicitly updated. The authors also offer a query language (Future Temporal Logic - FTL) based on temporal logic to formulate questions about the near future movement. The approach is restricted to moving points and does not address more complex time-varying geometries such as moving regions.

As an extension to the abstract model in [8], the concept of *spatio-temporal predicates* is introduced in [5]. The goal is to investigate temporal changes of topological relationships induced by temporal changes of spatial objects. Further work on modeling includes [18] where the authors focus on moving point objects and the inclusion of concepts of differential geometry (speed, acceleration) in a calculus based query language.

In contrast to the previous approaches, which offer limited temporal functionality, HERMES provides a full operative framework for the management of any temporal related type of data, as moving objects are. This is realized through the implementation of TLL [9] as a data cartridge [13]. TAU-TLL provides clear semantics for the time line including the time boundaries, time order, time reference, multiple temporal granularities, and the supported calendar. In addition, it provides an extensive set of object types and methods (superset of the corresponding ODMG [2]) for these temporal types.

In HERMES, the spatial functionality is provided by an OpenGIS-compatible ORDBMS as a separate data cartridge. The models proposed in the literature, provide separate objects for constructing different spatial geometries (e.g. points, lines, regions). In our case, we have a uniform representation of all kinds of geometries under the same spatial object, which increases the flexibility and the interoperability between moving types and pure spatial objects.

HERMES-MDC introduces a rich type system of time-varying geometries that change location or shape in discrete steps and/or continuously. An extensive set of object methods is developed that expresses all the interesting spatio-temporal phenomena and processes. This set of operations is a superset of the operations introduced in [8]. HERMES Type System introduces new objects like *Moving_Circle*, *Moving_Rectangular*, *Moving_Collection*, *Moving_Object*. The *Moving_Collection* object supports not only

a homogeneous collection of moving types but also heterogeneous. The *Moving_Object* can substitute any of the other moving types, as well as moving geometries that result as operations on other moving geometries and, moreover, it can model time-varying objects like the time-changing perimeter of a moving region. In [8], such degenerated moving types (e.g. moving real) are constructed as separate objects, leading to a unnecessary proliferation of object types. What is more, the flexibility of HERMES-MDC can be demonstrated in various ways. For example, moving linestrings that intersect themselves during their development are adopted by HERMES, while they are forbidden in [8]. Apart from linear interpolations of spatial and moving types utilized in [6], HERMES also utilizes arc interpolations. The expressiveness of the query language was established in Section 4.

6. CONCLUSIONS AND FUTURE WORK

In this paper, a data cartridge for moving objects, called HERMES-MDC, was introduced. This data cartridge is a system extension that provides spatio-temporal functionality to OpenGIS-compatible ORDBMS and supports modeling and querying of moving objects changing location either in discrete steps or continuously. To evaluate HERMES server-side MOD extensions we implemented a prototype application whose purpose is to provide a flexible means to apply benchmark queries proposed for the evaluation of systems providing location based services. Future work includes considering query optimization as well as indexing extensibility interfaces of current ORDBMS in order to enhance the performance of HERMES-MDC data cartridge, as well as knowledge discovery from MODs using data mining techniques.

7. ACKNOWLEDGMENTS

Research partially supported by the FP6-14915 IST/FET Project *GeoPKDD* (Geographic Privacy-aware Knowledge Discovery and Delivery) funded by the European Union and the *Pythagoras* EPEAEK II Programme of the Greek Ministry of National Education and Religious Affairs, co-funded by the European Union.

8. REFERENCES

[1] T. Abraham, J.F. Roddick. Survey of Spatio-Temporal Databases. *GeoInformatica*, 3:61-99, 1999.

[2] R.G.G. Cattel, D.K. Barry (eds.). *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, May 1997.

[3] S. Dieker and R. H. Guting, "Plug and Play with Query Algebras: Secondo. A Generic DBMS Development Environment", *Proc. Int'l Database Engineering and Applications Symposium* (IDEAS), 2000.

[4] M. Erwig, R.H. Güting, M. Schneider, and M. Vazirgiannis. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *GeoInformatica*, 3(3): 265-291, 1999.

[5] M. Erwig and M. Schneider. Spatio-Temporal Predicates. *IEEE Transactions on Knowledge and Data Engineering*, 14(4): 881-901, 2002.

[6] L. Forlizzi, R. H. Güting, E. Nardelli, M. Schneider. A Data Model and Data Structures for Moving Objects Databases. *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, Dallas, Texas, USA, 2000.

[7] S. Feuerstein and B. Pribyl. *Oracle PL/SQL Programming*. O'Reilly & Associates, 1997.

[8] R.H. Güting, M. H. Bohlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. *ACM Transactions on Database Systems*, 25(1): 1-42, 2000.

[9] I. Kakoudakis. *The TAU Temporal Object Model*. MPhil Thesis, UMIST, Department of Computation, 1996.

[10] M. Koubarakis, T. Sellis et al. (eds.). *Spatio-temporal Databases: The Chorochronos Approach*. Springer, 2003.

[11] J. A. C. Lema, L. Forlizzi, R. H. Güting, E. Nardelli, M. Schneider. Algorithms for Moving Objects Databases. *The Computer Journal* 46(6): 680-712, 2003.

[12] Oracle Corp. Oracle Database Documentation Library, 10g Release 1 (10.1), URL: http://otn.oracle.com/pls/db10g/. (accessed on 10 April 2006).

[13] N. Pelekis. *STAU: A Spatio-Temporal Extension to ORACLE DBMS*. PhD Thesis, UMIST, Department of Computation, 2002.

[14] N. Pelekis, Y. Theodoridis, S. Vosinakis, T. Panayiotopoulos. Hermes - A Framework for Location-Based Data Management. *Proc. 10th Int'l Conference on Extending Database Technology* (EDBT), LNCS 3896, Munich, Germany, 2006.

[15] N. Pelekis, B. Theodoulidis, I. Kopanakis, Y. Theodoridis. Literature Review of Spatio-Temporal Database Models. *The Knowledge Engineering Review journal*, 19(3), 235-274, June 2005.

[16] D. Peuquet. Making Space for Time: Issues in Spase-Time Data Representation. *GeoInformatica*, 5: 11-32, 2001.

[17] P. Sistla, O. Wolfson, S. Chamberlain, S.Dao. Modeling and Querying Moving Objects. *Proc. 13th Int'l Conf. on Data Engineering* (ICDE13), Birmingham, UK, 1997.

[18] J. Su, H. Xu and O. Ibarra. Moving Objects: Logical Relationships and Queries. *Proc. 7th Int'l Symp. on Spatial and Temporal Databases* (SSTD), Redondo Beach, California, USA, 2001.

[19] Y. Theodoridis. Ten Benchmark Database Queries for Location-based Services, *The Computer Journal*, 46(6):713-725, 2003.

[20] D. Pfoser, C. S. Jensen, and Y. Theodoridis, Novel Approaches to the Indexing of Moving Object Trajectories, *Proc. 26th Int'l Conf. on Very Large Data Bases*, Cairo, Egypt, 2000.

[21] O. Wolfson, A. P. Sistla, S. Chamberlain and Y. Yesha. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases*, 7 (3): 257-387, 1999.

[22] O. Wolfson, B. Xu, S. Chamberlain, L. Jiang. Moving Objects Databases: Issues and Solutions. *Proc. 10th Int'l Conf. on Scientific and Statistical Database Management*, Capri, Italy, 1998.

[23] P. Zhang. *The Spatial Movement Extensions of STAU*. MPhil Thesis, UMIST, Department of Computation, 2003.