HERMES: A Trajectory DB Engine for Mobility-Centric Applications

Nikos Pelekis (*) is a lecturer at the Department of Statistics and Insurance Science, University of Piraeus. Born in 1975, he received his B.Sc. degree from the Computer Science Department of the University of Crete (1998). He has subsequently joined the Department of Computation in the University of Manchester (former UMIST) to pursue his M.Sc. in Information Systems Engineering (1999) and his Ph.D. in Moving Object Databases (2002). He has been working for almost ten years in the field of data management and mining. He has co-authored more than 40 research papers and book chapters, he is a reviewer in many international journals and conferences. He has been actively involved in more than 10 European and National R&D projects. His research interests include data mining, spatiotemporal databases, management of location-based services, machine learning and geographical information systems.

Nikos Pelekis
University of Piraeus
Department of Statistics and Insurance Science
80 Karaoli-Dimitriou St., GR-18534 Piraeus, Greece
Tel: +30-2104142449, Fax: +30-2104142264
npelekis@unipi.gr

Elias Frentzos, received his Diploma in Civil Engineering and MSc in Geoinformatics, both from NTUA. He also holds a PhD from the Department of Informatics of the University of Piraeus where he is currently a Postdoc researcher, scholar of the Greek State Scholarships Foundation. He has published more than 20 papers in scientific journals and conferences such as IEEE TKDE, KAIS, Geoinformatica, ACM SIGMOD and IEEE ICDE. He has participated in several national and European research projects, and also involved in the development of several commercial GIS-related applications and projects. His research interests include spatial and spatiotemporal databases, location-based services and geographical information systems.

Elias Frentzos
University of Piraeus
Department of Informatics
80 Karaoli-Dimitriou St., GR-18534 Piraeus, Greece
Tel: +30-2104142449
efrentzo@unipi.gr

Nikos Giatrakos is a PhD Candidate at the Department of Informatics, University of Piraeus (Greece), supervised by Assoc. Professor Yannis Theodoridis. He received his Bachelor of Science degree (2006) in Informatics from the University of Piraeus and his Master of Science degree (2008) in Information Systems from the Athens University of Economics and Business (Greece). He has coauthored several journal and conference papers including ACM SIGMOD and IEEE ICDE. His research interests include distributed mining on data streams, mining spatiotemporal streaming data and trajectory warehousing. Since 2007, he has been involved in various national and European research projects.

Nikos Giatrakos
University of Piraeus
Department of Informatics
80 Karaoli-Dimitriou St., GR-18534 Piraeus, Greece
Tel: +30-2104142437
ngiatrak@unipi.gr

Yannis Theodoridis is Associate Professor with the Department of Informatics, University of Piraeus (UniPi), where he currently leads the Information Systems Lab (http://infolab.cs.unipi.gr). Born in 1967, he received his Diploma (1990) and Ph.D. (1996) in Electrical and Computer Engineering, both from the National Technical University of Athens, Greece. His research interests cover database management, geographical information management, data mining and knowledge discovery. Apart from several national-level projects, he was scientist in charge and coordinator of two European projects, namely PANDA (FP6/IST, 2001-04) and CODMINE (FP6/IST, 2002-03), on pattern-based management and privacy-preserving data mining, respectively, also participating in GeoPKDD (FP6/IST, 2005-09), MODAP (FP7/ICT, 2009-12) and MOVE (COST, 2009-12), on moving objects database management and knowledge discovery. He has or is serving as general co-chair for SSTD'03 and ECML/PKDD'11, vice PC chair for IEEE ICDM'08, member of the editorial board of the Int'l Journal on Data Warehousing and Mining (IJDWM), and member of the SSTD endowment. He has co-authored three monographs and over 80 refereed articles in scientific journals and conferences with more than 600 citations. He is member of ACM and IEEE.

Yannis Theodoridis
University of Piraeus
Department of Informatics
80 Karaoli-Dimitriou St., GR-18534 Piraeus, Greece
Tel: +30-2104142449
ytheod@unipi.gr

HERMES: A Trajectory DB Engine for Mobility-Centric Applications

Nikos Pelekis, University of Piraeus, Greece
Elias Frentzos, University of Piraeus, Greece
Nikos Giatrakos, University of Piraeus, Greece
Yannis Theodoridis, University of Piraeus, Greece

ABSTRACT

This paper presents HERMES, a prototype DB engine that defines a powerful query language for trajectory databases, which enables the support of mobility-centric applications, such as Location-Based Services (LBS). HERMES extends the data definition and manipulation language of Object-Relational DBMS (ORDBMS) with spatio-temporal semantics and functionality based on advanced spatio-temporal indexing and query processing techniques. Its implementation over two ORDBMS and its utilization in various domains proves the expressive power and applicability of HERMES in different application domains where knowledge regarding mobility data is essential. As a proof-of-concept, in this paper HERMES is applied to a case study related with vehicle traffic analysis, demonstrating its flexibility and usefulness for delivering custom-defined LBS.

Keywords: HERMES, Trajectories, Mobility, Molile Data Management, Location-Based Services.

INTRODUCTION

Moving Object Databases (MOD) (Güting et al., 2000) and more specifically Trajectory Databases (TD) being at the core of spatio-temporal database research, have emerged due to the explosion of mobile devices and positioning technologies. A MOD is the basic component of any mobility-centric application (Kargin, Basoglu, & Daim, 2009). However, although such LBS applications are already in the air for some years, the services currently provided are rather naive, not exploiting the current software capabilities and the recent advances in MOD research field. We argue that one of the reasons for this is due to the common practice in existing approaches, which provides services to mobile users by just taking into account the current location-time and velocity information, arriving at the MOD server as a sequence of updates. Given this model and the fact that LBS applications need to handle huge volumes of data, it rationally arises that performance is a significant problem; therefore, efficient query processing and indexing techniques should be applied. Moreover, this model has limited applicability in real-world applications, since safe estimations about future positions should involve past positions as well.

The key observation that motivates HERMES is that the more the knowledge in hand about the trajectory of a mobile user, the better the exploitation of the advances in spatio-temporal query processing for providing intelligent LBS. Based on this motivation, the aim of this paper is to describe a robust framework capable of aiding either an analyst working with mobility data, or more technically, a developer who models, queries a TD and builds a mobility-centric application on top of the TD.

Moreover, given the ubiquitousness of location-aware devices, databases handling moving objects will, sooner or later, face enormous volumes of data. It consequently arises that performance in the presence of vast data sizes, is a significant problem for moving object databases and the only way to deal with such enormous sizes is the exploitation of specialized access methods used for spatio-temporal indexing purposes. The domain of spatio-temporal indexing, as well as other related domains, such as multimedia (Chatterjee, & Chen, 2010) and spatial indexing, is dominated by the presence of the R-tree, along with its variations and extensions. Among others, 3D R-trees (Theodoridis, Vazirgiannis, & Sellis, 1996), TB-trees and STR-trees (Pfoser, Jensen, & Theodoridis 2000), and PA-trees (Ni, & Ravishankar, 2007) are considered as extensions of the R-tree in the spatio-temporal domain. As in the case of appropriate moving object data types and methods for extending the type system of ORDBMS, except the well-known R-trees, which are suitable only for static spatial data, none of the above proposals have been incorporated into existing ORDBMS. Among them, the Trajectory Bundle tree (TB-tree) (Pfoser et al., 2000), is adopted in this work and appropriately designed and implemented inside HERMES taking advantage of the indexing extensibility interface of ORDBMS. Being a member of the R-tree family, TB-tree is able to support traditional queries such as range and distance-based queries. At the same time, it supports objects moving on the unconstrained space, and is the only one that fulfills the need for trajectory preservation so as to efficiently support trajectory-based operations.

Furthermore, apart from simple query operators (e.g. range queries) natively supported by R-trees, there is a variety of spatio-temporal operators which require more sophisticated query processing techniques in order to be efficiently processed. Among them, an important class of queries is the so-called k nearest neighbor (k-NN) search, where one is interested in finding the k closest trajectories to a predefined query object Q (stationary or moving). Thus, one of the challenges being present in the domain of trajectory databases is to develop mechanisms to perform k-NN search on MODs exploiting spatio-temporal indexes storing historical information. Among the solutions proposed in the literature we adopt the one proposed by (Frentzos, Gratsias, Pelekis, & Theodoridis 2007) which efficiently supports Nearest Neighbor (NN) queries over historical trajectory data.

Finally, as we aim at providing a powerful toolkit for analysts, HERMES provides qualitatively different techniques for trajectory similarity search, which is exploited to support trajectory clustering and classification mining tasks that imply a way to quantify the distance between two trajectories. More specifically, we adopt a novel set of trajectory distance functions (Pelekis, Kopanakis, Ntoutsi, Marketos, Andrienko & Theodoridis, 2007; Pelekis, Andrienko, Andrienko, Kopanakis, Marketos, & Theodoridis, 2010) based on primitive (space and time) as well as derived parameters of moving objects (speed, acceleration, and direction), which are also capable to support sub-trajectory similarity matching. The overall framework advances the contribution of our approach by two inter-related facts: firstly, the combination of the similarity operators in the extended with MOD semantics SQL-like query language (using AND/OR clauses) provides analysis functionality unmatched so far (e.g. "find objects that moved closely in space but with very dissimilar speed patterns"); secondly, the output of each of the supported operators defines similarity patterns that can be utilized to reveal local similarity features (e.g. "find the most similar portions between two, in general, dissimilar trajectories").

Summarizing the previous discussion, the contributions of the paper are the following:

- We present a datatype-oriented model and a SQL-like query language for supporting TD on top of OGC-compliant ORDBMS, while we describe the architecture of our server-side TD engine and the interface for building advanced mobility-centric applications.
- We demonstrate how novel, appropriate access methods and advanced, non-trivial query operators are embedded inside extensible ORDBMS providing efficiency and higher level analysis functionality.
- We investigate the expressive power and flexibility of the produced query language via a real-world application scenario.
- As a proof of concept, we have implemented the proposed framework on top of a commercial ORDBMS, namely Oracle, while our design has also been successfully applied and repeated in the open-source PostgreSQL with the PostGIS spatial extension (Boulahya, 2009).

To the best of our knowledge, HERMES is the first work that presents a complete set of state-of-the-art query processing algorithms for TD, which has been incorporated into state-of-the-art OGC-compliant ORDBMS.

The outline of the paper is as follows: we first present the data type system for TD introduced in HERMES and then, we discuss implementation aspects. A representative set of methods that extend the query language of an ORDBMS with spatio-temporal semantics is then discussed. Subsequently, the architecture for implementing HERMES in a state-of-the-art ORDBMS is presented, while a proof-of-concept case study related with vehicle traffic analysis follows. We assess the applicability of the proposed system in building other systems via presenting four tools and corresponding application domains that utilize HERMES as the platform for managing and analyzing their movement related data. Finally, we conclude the paper, also pointing out some interesting future research directions.

A DATA TYPE MODEL FOR TRAJECTORY DATABASES

Preliminaries of Trajectory Data Types

In order to define a data type model for TD, we need to base on standard database types built into any DBMS, as well as temporal and spatial types.

Temporal types are introduced by *TAU Temporal Literal Library* (*TAU-TLL*) in (Pelekis, 2002), which is the component of HERMES system responsible for providing pure temporal object-relational functionality. *TAU-TLL* implements the *Time Model*, adopted by the *TAU Temporal Object Model*, and augments the four temporal literal data types found in *ODMG* object model (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, spatial types (point, line segment, rectangle, etc.) are supported by an *OGC Geometry* (i.e. a spatial type that conforms to the specifications of the Open Geospatial Consortium). Such a spatial extension is found in several state-of-the-art

ORDBMS (e.g. DB2 spatial extender, MySQL spatial extension, Oracle, Postgis, SQL Server) and provides an integrated set of functions and procedures that enable spatial data following the OGC standard to be efficiently stored in a spatial database, accessed and futher processed. Of course, the geometric operations forming the behavior of spatial types supported by these extensions, handle queries statically, meaning that there exists no notion of time associated to the spatial objects. This is exactly the target addressed in the type system for trajectories that we propose in the sequel.

In (Güting et al., 2000; Forlizzi, Güting, Nardelli, & Schneider, 2000; Lema, Forlizzi, Güting, Nardelli, & Schneider, 2003) the authors introduce the concept of *sliced representation*, the basic idea of which is to decompose the temporal development of a moving value into fragments called *"slices"* such that within the slice this development can be described by some kind of *"simple"* function. This is illustrated in Figure 1 for a time-varying point (moving point).

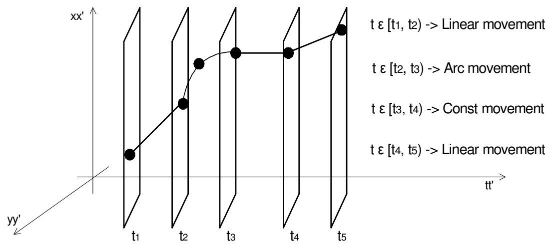

Figure 1 Moving Point with various types of movement

In this work, we adopt and extend the *sliced representation* concept and utilize it in the implementation of the proposed type system. In order to use the sliced representation to define a moving point (i.e. trajectory), one has to decompose the definition of a moving point into several definitions, one for each of the slices that corresponds to a simple function, and then compose these sub-definitions as a collection to define the moving point. Each one of the sub-definitions corresponds to a so-called *unit moving point*.

In order to define a unit moving point, we need to associate a period of time with the description of a simple function that models the behavior of the moving point 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 *simple function*. The first concept has been already introduced as one of the temporal literal types in *TLT* (closed-open *period*). 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.

In the case of arc motions, following the categorization of realistic arc motions initially discussed in (Zhang, 2003), 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).

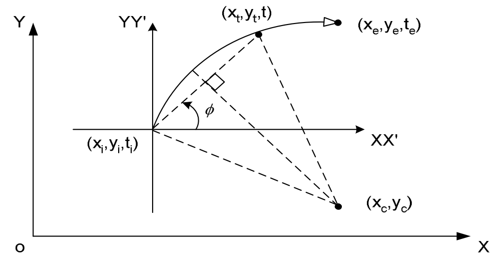

Figure 2 Motion on a circular arc

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.

$$\begin{aligned}
&ARC_1(t) \Rightarrow (x_t, y_t) = (x_t + L_t \times Cos\phi,\ y_t + L_t \times Sin\phi) \\
&L_t = 2 \times R \times Sin\left(\frac{S_t}{2 \times R}\right) \\
&S_t = v \times t + \frac{1}{2} \times a \times t^2 \\
&\phi = \frac{\pi}{2} + Sin^{-1}\left(\frac{y_c - y_i}{R}\right) - \frac{S_t}{2 \times R} \\
&v \ge 0,\ t \in [t_i, t_e),\ \phi \in (-\infty, +\infty)
\end{aligned} \quad (1)$$

Consequently, in the general case the Unit_Function is defined as follows:

$$\begin{aligned}
&Unit_Function =_d \langle\langle\ x_i{:}double,\ y_i{:}double, \\
&\quad x_e{:}double,\ y_e{:}double,\ x_c{:}double,\ y_c{:}double, \\
&\quad v{:}double,\ a{:}double,\ flag{:}TypeOfFunction \rangle\rangle
\end{aligned} \quad (2)$$

where *ITTypeOfFunction T={PLNML_1, ARC_*

<1..8>, CONST }, meaning 1st order polynomial, one of the eight possible circular arcs, and constant function, respectively.

Combining *time period* and *unit function* together, the primitive *Unit_Moving_Point* is defined. This is a fundamental type since it represents the smallest granule of movement of a trajectory. Formally:

$$Unit_Moving_Point =_d \langle p: period\langle SECOND\rangle, m: Unit_Function\rangle \quad (3)$$

We now introduce the moving point type that plays the dominant role in our spatio-temporal data type system. The process that we followed to define the moving point is to introduce it as a collection of the corresponding unit moving point, which means, in terms of object orientation, that there exists a composition relationship between the unit moving point and the moving point. As such, the *Moving_Point* object type is introduced as a collection of *Unit_Moving_Point* object types. Formally:

$$Moving_Point =_d \{ p: set\langle Unit_Moving_Point\rangle \mid \forall i, j \in ulong, 1 \le i, j \le |set\langle Unit_Moving_Point\rangle| : j = i+1 \Rightarrow p_i.p < p_j.p \wedge \neg overlaps(p_i.p, p_j.p) \wedge \forall t \in double: inside(t, p_i.p) \Rightarrow at_instant(p, t) \in OGC\text{-}GEOMETRY_{GTYPE}{=}point \} \quad (4)$$

In the following two sections we describe the indexing capabilities of HERMES on the above defined moving point data type, while we introduce the corresponding prerequisite data types.

Spatiotemporal Indexing in HERMES

In this section we briefly introduce the basic notions of spatio-temporal indexing and present the TB-tree which is adopted in this work and implemented in HERMES. Similar to the original R-tree, the TB-tree is a height-balanced tree with the index records in its leaf nodes; leaf nodes contain entries of the same trajectories, and are of the form $S = \langle MBB, Orientation\rangle$, where MBB is the 3D bounding box of the 3D line segment belonging to an object's trajectory (handling time as the third dimension) and *Orientation* is a flag used to reconstruct the actual 3D line segment inside the MBB among four different alternatives that exist (see Figure 4). Moreover, contrary to the well-known B-tree, and similarly to the original R-tree, internal and leaf node MBBs belonging to the same tree level are allowed to overlap. Each internal or leaf node in the tree corresponds to a physical *disk page* (or disk block, which is the fundamental element on which the actual disk storage is organized) and contains between *m* and *M* entries (*M* is the node capacity and *m* in the case of TB-tree is set to 1).

Since each leaf node contains entries of the same trajectory, the object *id* can be stored once in the leaf node header. Therefore, TB-tree leaf nodes are of the form $\langle header, \{S_i\}\rangle$, where each $S_i = \langle MBB_i \ Orientation_i\rangle$ and header = $\langle id, \#entries, ptrCur, ptrParent, ptrNext, ptrPrevious\rangle$ (in other words, the object identifier, the number of node entries and four pointers, to the current, the parent, and the next and previous nodes of the same trajectory). On the other hand, non-leaf nodes are of the form $\langle header, \{E_i\}\rangle$, where each $E_i = \langle MBB_i, ptr_i\rangle$ with MBB_i be the enclosing 3D box of the child node pointed by ptr_i (a pointer to it), and *header* = $\langle \#entries, ptrCur, ptrParent\rangle$ simply stores the number of node entries and a pointer to itself and to its parent node. Furthermore, similar to SETI (Chakka, Everspaugh, & Patel., 2003) and in order to support high insertion rates, our TB-tree implementation uses an in-memory hashed front-line structure, which maintains tuples of the form $\langle id, P_{curr}, N_{curr}\rangle$ with the object identifier *id*, its latest position $P_{curr} = \langle t_{curr}, x_{curr}, y_{curr}\rangle$ and a pointer N_{curr} to the leaf node containing P_{curr}.

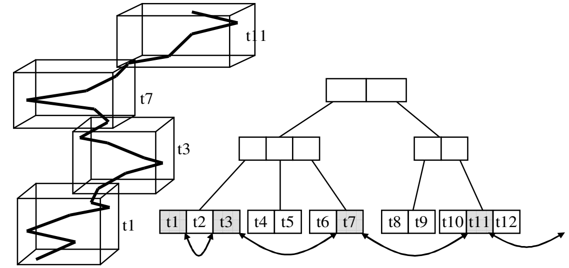

Figure 3 The TB-tree structure

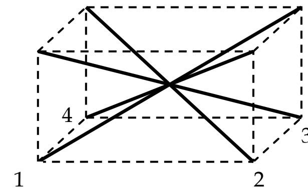

Figure 4 Alternative ways that a 3D line segment can be contained inside a *MBB*

Given the size of a disk block, which is predetermined by the operation system, the number of elements contained in a leaf of internal node in the tree is resticted by it. Specifically, given that each S_i is contained in 25 bytes (4 bytes for each one of the 6 double precission numbers needed to describe the MBB and 1 byte for the orientation flag) and the header of each leaf node has the size of 16 bytes (4

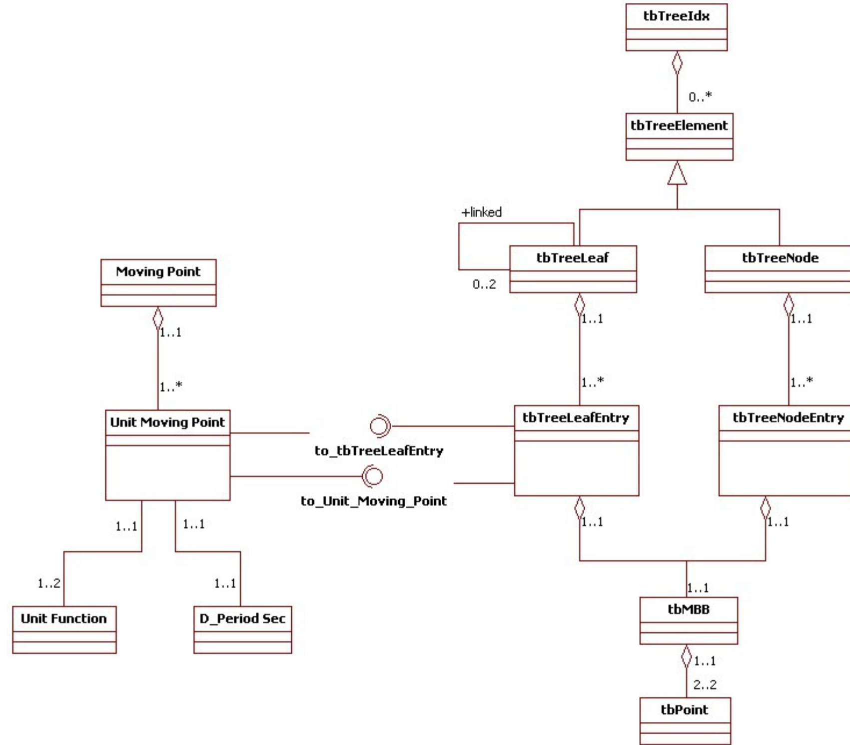

Figure 5 TB-tree data types

bytes for each one of the object identifier, the number of entries, and the four pointers), the total leaf capacity in terms of trajectory segments is given by *Int(([page size]*-16*)/*25*)*; this number for a typical page size of 4096 bytes results in 163 trajectory segments inside each leaf node. Following the same reasoning each internal tree node has a capacity of 170 entries, resulting in 170 child nodes.

The difference of the TB-tree with the majority of the R-tree variations relies on the way the index is built. Specifically, its insertion algorithm is not based upon the spatial and temporal relations of moving objects (or moving object segments) but it relies only on the moving object identifier (*id*). When new line segments are inserted, the algorithm searches for the leaf node containing the last entry of the same trajectory, and simply inserts the new entry in it, thus forming leaf nodes that contain line segments from a single trajectory. Furthermore, its split strategy is very simple: when a leaf node is full, a new one is created and is inserted at the right-end of the tree; due to the monotonicity of time, this strategy ensures that trajectories are organized monotonically inside the tree structure, e.g., trajectory segments are organized by time. For each trajectory, a double linked list connects the leaf nodes that contain its portions together (Figure 3), resulting in a structure that can efficiently answer trajectory-based queries.

The TB-tree Data Types

In this section we introduce the data types required for embedding the TB-tree in an ORDBMS that supports moving objects. We should note that these data types are transparent to the user of HERMES and their usage is just for the internal construction of the tree. The implementation of a tree-based index under the object-relational model follows a number of well-known rules and techniques, such as implementing different object classes for each one of the basic tree elements, namely, tree nodes (root, internal nodes, leafs) and node elements.

Figure 5 provides an abstract, though insightful, view of the index organization, along with the connection with the rest of the HERMES data types in the form of a UML class diagram describing the structure's primitives. The left part of the diagram depicts the objects participating in the index formation. Following a top-down description, the *tbTreeIdx* class is used mainly for completeness as an abstraction of the corresponding part of the model and it refers to the definition of TB-tree index on the table where the actual trajectory data are stored. Since the main trajectory table may initially be empty, the corresponding aggregation with the lower-level *tbTreeElement* class possesses a cardinality of «0..*».

Descending the diagram, we observe that the

whole arrangement is separated in two kinds of *TB-tree Node* types. Namely, the *tbTreeNode* Class regarding the internal nodes of the tree structure and the *tbTreeLeaf* class used to represent the leaf nodes of the index where trajectory segments are stored. Given that the size of each leaf node is predetermined and equivalent to the chosen disc block size, its capacity in terms of trajectory line segments is also predefined (following the previous discussion, a page size of 4096 bytes results in leaf nodes fitting no more than 163 segments). As a consequence, exceeding the aforementioned size, in terms of leaf node entries, causes segments of the same trajectory to be stored in different leaf nodes which remain connected by means of a double linked list. This is denoted using an association termed as *"linked"*. Note that the head leaf of the list might be connected to at most 1 (or 0 when the trajectory fits in a block) other leaves and the same holds for the tail of the arrangement. Each intermediate node is essentially linked to two other peers. This explains the cardinality of the respective association.

A *tbTreeLeaf* includes a number of leaf entries (*tbTreeLeafEntry* in Figure 5), each consisting of the MBB (*tbMBB* in the figure) that surrounds the trajectory segment kept in the leaf entry, along with an integer number 1-4 denoting its orientation; *tbMBB*s is composed by a *MinPoint* and a *MaxPoint* of *tbPoint* type which are the lower left and upper right of the box, respectively in the spatio-temporal space, while *tbpoint* has only a property of *tbX* collection type, which is an array of size 3 used to hold triplets (x,y,t) of time-stamped positions forming the entire object's trajectory. More specifically, the attributes of *tbTreeLeaf* are:

- *MoID* of integer type which is the global trajectory identifier,
- *ptrCurrentNode* of integer type, being the current node's identifier encapsulated in the object to facilitate implementation issues,
- *ptrParentNode* of integer type, representing a pointer to the parent of the current node used to ascend the tree when necessary,
- *ptrPreviousNode* of type integer, which is a pointer to the node containing the previous fragment of the same trajectory,
- *ptrNextNode* of type integer, which is a pointer to the node containing the next fragment of the same trajectory,
- *LeafEntries,* a collection of *tbTreeLeafEntry* type with fixed capacity, which involves the current leaf entries as previously described, and,
- *count* of integer type that holds the cardinality of *LeafEntries*.

Formaly, given the leaf capacity *LeafCapacity*, i.e., the maximum number of leaf entries that may be contained in a leaf node, we define the following types:

$$tbPoint=_d\{tbX:set\langle double\rangle \mid |tbX|=3\} \quad (5)$$

$$tbMBB=_d\{\langle MinPoint:tbPoint, MaxPoint:tbPoint\rangle \mid \forall 0\le i\le 2, MinPoint.x(i)<=MaxPoint.x(i)\} \quad (6)$$

$$tbTreeLeafEntry=_d\{\langle MBB:tbMBB, Orientation: short\rangle \mid Orientation<4\} \quad (7)$$

$$tbTreeLeaf=_d\{\langle MovingObjectId:long, ptrCurrentNode:long, ptrParentNode:long, ptrNextNode:long, ptrPreviousNode:long, LeafEntries: set\langle tbTreeLeafEntry\rangle, count:long \mid |LeafEntries|\le LeafCapacity, count=|LeafEntries|\rangle\} \quad (8)$$

Similarly, a *tbTreeNode* contains a set of *tbTreeNodeEntry* objects; each *tbTreeNodeEntry* encloses all the leaf or node entries contained in the sub-tree starting with this node as root. More spectifically, its attributes involve:

- *ptrCurrentNode* of integer type, which is the current node's identifier encapsulated in the object to facilitate implementation issues,
- *ptrParentNode* of integer type, which is a pointer to the parent of the current node used to ascend the tree when necessary,
- *NodeEnties,* a collection of *tbTreeNodeEntry* type with fixed capacity, which involves the current node entries as previously described, and,
- *count* of integer type to hold the cardinality of *NodeEntries*.

Formally, given the node capacity *NodeCapacity* we define:

$$tbTreeNodeEntry=_d\{\langle MBB:tbMBB, ptr: long\rangle\} \quad (9)$$

$$tbTreeNode=_d\{\langle ptrParentNode:long, ptrCurrentNode:long, NodeEntries: set\langle tbTreeNodeEntry\rangle, count:long \mid |NodeEntries|\le NodeCapacity, count=|NodeEntries|\rangle\} \quad (10)$$

Eventually, the two interfaces of Figure 5 *to_tbTreeLeafEntry*, *to_Unit_Moving_Point* provide essential mechanisms for object transformation from one type to the other.

The following sections describe the design decisions and the implementation details for mapping the *TD* type system into extensible ORDBMS, as well as essential functionality for extending SQL-like query languages with *TD* querying constructs.

ON THE PHYSICAL IMPLEMENTATION OF THE HERMES TD TYPE SYSTEM

The physical representation of the data types reflects the structures that are necessary in order to capture the semantics and implement the methods of these data types. In this section, we discuss how the types abstractly described in the previous section are mapped to physical structures for storing them into an ORDBMS with an OGC-compliant spatial extension.

Implementation of the Primitive Data Types

The primitive data types of HERMES are the *Unit_Function*, the *Unit_Moving_Point* and the *Moving_Point*. *Unit_Function* is constructed as an octave of real numbers and a flag indicating the type of the simple function. In the current version, three types of functions are supported, namely polynomial of first degree, circular arc and the constant function. The modeling of *Unit_Function* is extensible; for example, if one wishes to add interpolations with spline or polynomials with degree higher than one, then what is only needed to be done is the addition (if necessary) of the appropriate variables as attributes of the object and the implementation of such a function.

We should note that we model a moving point that changes discretely for a period of time by setting all *Unit_Function* objects of the corresponding unit moving point to be constant functions. Due to the fact that the coordinates represented by these *Unit_Function* objects do not change for this period of time, it is equivalent to taking a snapshot of the moving point, which is valid for the entire period. If at least one of these unit functions is not constant then the moving point is continuous for this period of time.

Finally, we construct a *Moving_Point* object type as a collection of *Unit_Moving_Point* objects (i.e. pointer to a nested table or a varying length array (i.e. varray), depending on the underlying ORDBMS, of *Unit_Moving_Point* objects), which in turn are defined as objects consisting of two attributes. The first attribute is the time period during which the other attribute is defined. The time period is expressed as an open-closed *Period* object, while the other attribute is of *Unit_Function* object type, whose domain of definition is the set of real numbers inside the open interval $[t_1, t_2)$, where t_1 is the starting point of the period and t_2 is the ending point of the period.

Implementation of the TB-tree Data Types

Regarding the data types required for the TB-tree index, they are mainly implemented as objects with simple attributes and arrays of attributes. Specifically:

- *tbPoint* is constructed as a standard array of real values with its cardinality set to 3 (x, y and t)
- *tbMBB* is constructed by two attributes of type *tbPoint*
- *LeafEntry* is constructed by an attribute of *tbMBB* type and another one of integer type taking values from 1 to 4, representing one among the four possible orientations that a line segment may have inside its MBB.
- *tbTreeLeaf* is constructed by the integer value of *MovingObjectId*, and a set of pointers (integer values), i.e., *ptrCurrentNode, ptrParentNode, ptrPreviousNode* and *ptr NextNode.* It also contains a standard array of *tbTreeLeafEntries* with predetermined size *LeafCapacity*, and an integer value containing the number of occupied entries inside the array.
- Similarly, a *tbTreeNode* is constructed by the two pointers (integer values), *ptrCurrentNode* and *ptrParentNode,* and a standard array of *tbTreeNodeEntries* with predetermined size *NodeCapacity*. Finally, an integer value containing the number of occupied entries inside the aforementioned array is employed inside the *tbTreeNode* structure.

Regarding the implementation of the TB-tree in the HERMES a number of tables constituting the primary storage elements of index data are employed. Specifically, following the UML of Figure 5, the basic data types are stored in the following tables which are automatically created/dropped upon the respective index creation/drop:

- *movingobjects*: The *movingobjects* is an auxiliary table used to store a pointer to the index leaf where the last part of a trajectory is stored (Frentzos, 2008). As such, it contains only 2 columns for the *trajectory id*, and for the pointer integer values.
- *tbTtreeidx_non_leaf*: This is the table storing the internal tree nodes. It actually contains tuples of the form (*NodeId*, *tbTreeNode*), where *NodeId=tbTreeNode.ptrCurrentNode*.
- *tbTreeidx_leaf*: This is the table storing the tree leaf nodes; it also contains tuples of the form (*LeafId*, *tbTreeLeaf*) where *LeafId* = *tbTreeLeaf.ptrCurrentNode*.

EXTENDING HERMES WITH OBJECT METHODS AND OPERATORS

In this section, we present the operations of the moving types introduced by HERMES classified into appropriate categories that enable us to describe and analyze the new query capabilities. The identifiable classes of operations that HERMES supports are:

i) *Predicates and projection operations*: operations that return boolean values concerning topological and other relationships (e.g. intersection, within distance, etc.), operations that restrict and project moving types to temporal (e.g. at_instant, at_period) and spatial domain (e.g. trajectory).
ii) *Numeric operations*: functions that compute a numeric value (e.g. speed).
iii) *Distance functions*: a set of trajectory distance functions based on primitive (space and time) as well as derived parameters of trajectories (speed, acceleration, and direction).
iv) *Index maintenance*: necessary operations for creating, dropping and updating the TB-tree index.
v) *Index operators*: several advanced algorithms for efficient query processing of movement data.

The following sections describe the functionality of selected operations, representative of each class. The interested reader may find more operations in (Pelekis, & Theodoridis, 2010).

Predicates and projection operations

HERMES provides a rich palette of object methods of special interest to describe relationships between moving types that have been proposed in the literature. Subsequently, we present the operations and the semantics behind these methods. Most of these operations come with two different overloaded signatures, modeling different semantics: the first signature is time-dependent, meaning that the outcome of the operation is related to a user-defined time point, while the second is time-independent. For simplicity herein we present only one of the two, i.e. the most interesting. Also, many HERMES object methods accept a *tolerance* parameter which is usually a reflection of how accurate or precise users perceive their spatio-temporal data to be.

Op1. *boolean* ***f_within_distance*** *(distance, Moving_Point, tolerance, Timepoint):* This predicate determines whether two moving points are within some specified Euclidean distance from each other at a user-defined time point. After mapping the moving points to spatial points at the given instant, the function returns *TRUE* for object pairs that are within the specified distance; returns *FALSE* otherwise.

Op2. *Unit_Moving_Point* ***unit_type*** *(Timepoint):* This operation identifies (and returns) the unit-moving point whose attribute time period (*Period* object) *"contains"* the user-defined time point (*Timepoint* object). In other words, it returns that unit-moving point where the time instant represented by the argument *Timepoint* object is *"inside"* the time period that characterizes the unit-moving point.

Op3. *Geometry* ***at_instant*** *(Timepoint):* The *at_instant* operation maps the mathematical description of a unit function object (see formulas 1) to a spatial point object, where the moving point resists at the given timepoint.

Op4. *Moving_Point* ***at_period*** *(Period):* The *at_period* object method is an operation that restricts the moving point 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.

There are more operations for performing similar type of projections, like the *at_point* and *at_linestring* methods that either restricts a moving point to a static point or linestring geometry, respectively, or return the temporal point or period that the restriction is valid.

Op5. *Geometry* ***f_trajectory*** *():* This function simulates the route traversed by a *Moving_Point*. More specifically, this projection of the movement of a *Moving_Point* to the Cartesian plane is done by mapping the time-dependent coordinates of the object at the sampled time instants of the *Unit_Moving_Point* objects that compose the *Moving_Point*. Figure 6 illustrates this operation.

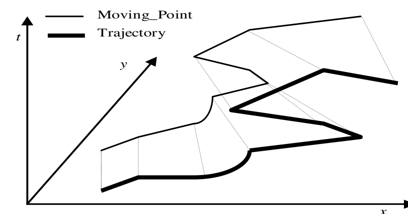

Figure 6 The trajectory of a *Moving_Point*

Op6. *Moving_Point* ***f_intersection*** *(Geometry, tolerance):* The *f_intersection* object method returns takes as a parameter a polygon geometry and returns the portion of the moving point inside the given region.

Numeric operations

HERMES supports a special class of object methods that compute either a numeric value or quantify a property related with the rate of change of a trajectory, as moving points are time-dependent objects. More specifically, we provide the subsequent operations:

Op7. *number* ***f_length*** *(tolerance, Timepoint):* The *f_length* object method computes the length of the route of a *Moving_Point* when projected at the Cartesian plane at a user-defined time point.

Op8. *number* ***f_speed*** *(Timepoint)*: The *speed* operation returns a number representing the *speed* of a moving point at a specific timepoint. The algorithm that implements the *speed* method is based on the time derivative of the distance function as described by Formula 1. This function is extracted from the

Unit_Function object of the corresponding *Unit_Moving_Point*.

Op9. *number* ***f_direction*** *(Moving_Point, Timepoint):* The *f_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 computed angle is the one formed by the conceptual line segment that joins the two points and the *xx'* axis.

Based on the above method HERMES supports two sets of operations that provide predicate functionality on directional relationships between moving objects. The first set consists of four operations (namely, *f_west*, *f_east*, *f_north*, and *f_south*) each of which returns a Boolean value depending on whether the moving object is e.g. *west* from the a given moving or static point parameter, as well as a range of angles that puts some constraints in the directional relationship. Similarly, the second set consists of four operations (namely, *f_left*, *f_right*, *f_above*, and *f_behind*) that represent implicit directional relationships w.r.t. the motion of the query object.

Distance functions

HERMES supports a set of query operators for similarity search between moving points as these have been introduced in (Pelekis et al., 2007). Two main types of similarities are defined, namely, *spatiotemporal* and (temporally-relaxed) *spatial* similarity, followed by three variations, namely *speed-pattern based*, *acceleration-pattern based*, and *directional* similarity. More specifically:

Op10. *number* ***GenLIP****(Moving_Point):* The *Generalized Locality Inbetween Polylines* (GenLIP) distance between two moving points, returns an intuitive value that implies the area (see the shaded area in Figure 7) between the spatial projections of the two trajectories.

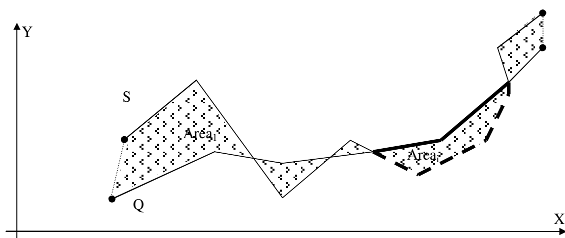

Figure 7: Locality In-between 2D Polylines

Op11. *number* ***GenSTLIP****(Moving_Point):* The *Generalized SpatioTemporal LIP* (GenSTLIP) function takes into account time, it operates on the original 3D representation of moving points and as such eliminates the time-relaxation of the GenLIP method by requiring co-location and co-existence during the lifetime of the moving points.

Op12. *number* ***GenSPSTLIP****(Moving_Point):*

Op13. *number* ***GenACSTLIP****(Moving_Point):* The *Generalized Speed-Pattern and Acceleration-Pattern STLIP* functions take also into account whether the two involved moving points move with similar speed (*GenSPSTLIP*) or acceleration (*GenACSTLIP*) patterns.

Op14. *number* ***DDIST****(Moving_Point):*

Op15. *number* ***TDDIST****(Moving_Point):* The *Directional Distance* (DDIST) and *Temporal DDIST* (TDDIST) are two other variations that quantify the similarity of two moving objects according to their heading pattern. The first variation operates on the spatial projection of the objects, while the second checks whether the change in the heading happens in a synchronized way.

4.4 Index maintenance

Based on the extensible indexing capabilities provided by an ORDBMS each TB-tree owns the following functions:

Op16. IndexCreate: creates the index tables (i.e *tbtreeidx_leaf*, *tbtreeidx_non_leaf*) and populates the data already inserted in the table on which the index is created.

Op17. IndexInsert: performs insertions in the tree, triggered by the insertion of a new trajectory on the indexed table.

Op18. IndexUpdate: updates the tree every time a new trajectory segment (i.e *Unit_Moving_Point*) is inserted.

Op19. IndexDrop: drops the tables that store the index data. This method is called when a DROP INDEX statement is issued.

Functions *IndexInsert* and *IndexUpdate* call function *TBINSERT* which implements the TB-tree's insertion algorihm as described in (Pfoser et al., 2000).

Index operators

Range/timeslice queries, of the form "*find all objects located within a given area during a certain time interval or time instance*", (*Q2*/*Q*1 in Figure 8), is a straightforward extension of the respective 2D R-tree algorithm, in the 3D space formed by the two spatial and the one temporal dimension. This algorithm recursively visits tree nodes, rejecting node *MBB*s that does not overlap the query window, while following the pointers from overlapping *MBB*s to their respective child nodes until all candidate leaf nodes have been found. The algorithm starts by visiting the tree root, checking whether the *MBB*s of the root entries overlap the spatio-temporal query window *Q*. If a node entry overlaps *Q*, the algorithm follows the pointer to the corresponding child node, where it re-

peats recursively the same task. If the algorithm reaches a leaf node, leaf entries are examined against *Q* and if their *MBB* overlap, the algorithm reports their *ids*.

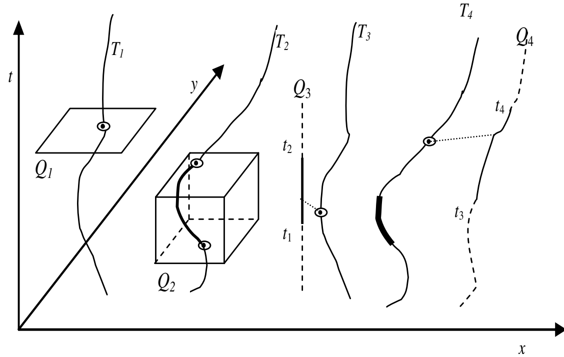

Figure 8 Querying trajectory databases

Regarding the *k* nearest neighbor (*k*-NN) search, (Frentzos et al., 2007) proposed a variety of solutions for answering such queries. More specifically, given as an example the trajectory database of Figure 8, given a stationary (or moving) query point *Q*3 (*Q*4) and a temporal query window [t_1, t_2] ([t_3, t_4]), (Frentzos et al., 2007) proposed several algorithms for finding the moving object trajectory T_3 (T_4) that is closer to the query object. Among them, the incremental varations of the algorithms proposed in (Frentzos et al., 2007) (*IncPointNNSearch* and *IncTrajectoryNN-Search*) are shown to be more scalable, thus, being good solutions to be implemented in the HERMES. Here, we have also to point that the aforementioned algorithms are capable to answer *k*-NN versions of the respective queries as well.

More specifically, the algorithms proposed in (Frentzos et al., 2007) traverse the tree structure in a best-first way. The proposed algorithms use a priority queue, in which the (node or leaf) entries of the tree nodes are stored in increasing order of their distance from the query object. At each tree node the algorithm iterates through its entries checking whether the lifetime of an entry overlaps the time period of the query, calculating at the same time its distance from the query object, which is used to store them in the priority queue. At each algorithm's iteration the first entry is requested from the queue, until a leaf entry is found, which is then reported as the query result. The algorithms proposed in (Frentzos et al., 2007) are incremental in the sense that the *k*-th NN can be obtained with very little additional work once the (*k*-1)-th NN has been found; therefore, are easily generalized to the case where are the *k*>1 nearest neighbors of a query object (stationary of moving point) are requested. Given the above discussion, HERMES supports the following set of operators, namely, *range* (Pfoser et al., 2000), *Point* and *Trajectory Nearest Neighbor* (Frentzos et al., 2007) and spatio-temporal *topological* (Pfoser et al., 2000) queries.

Op20.*Multi_Moving_Point* ***tb_mp_in_spatio temporal_window****(Geometry, Period)*: This function executes a range query against a table storing indexed trajectories. It takes as arguments a standard spatial window; therefore, *geometry* is restricted to rectangle schemes, as well as a temporal period, and returns trajectory a *Multi_Moving_Point* consisting of all trajectory fractions fully contained inside the given spatio-temporal window.

Op21. *set⟨integer, Unit_Moving_Point⟩* ***IncPointNNSearch*** *(Geometry, Period, k)*: This function executes a Point Nearest Neighbor query against a table storing indexed trajectories. It takes as arguments a query point, a temporal period, and the number of *k* closest nearest neighbors to be returned. It returns trajectory ids, as well as the respective *Unit_Moving_Point* that is closest to the query point at the given time period.

Op22. *set⟨integer, Unit_Moving_Point⟩* ***IncTrajectoryNNsearch*** *(identifier, k)*: executes a trajectory Nearest Neighbor query against a table storing indexed trajectories. It takes as arguments the *identifier* of the trajectory to be used as query, and the number of *k* closest nearest neighbors to be returned. It returns trajectory ids, as well as the respective *Unit_Moving_Point* that is closest to the query trajectory during its life time.

Op23. *set⟨integer⟩* ***TopologicalQuery****(Geometry, Period, mask)*: This function is used to retrieve the trajectories that enter and/or leave a spatio-temporal query window. The query parameters involve the geometry of a (rectangle) area, a time period and a *MASK* that declares the type of topological query. Possible *MASK* (string) values can be '*ENTER*', '*LEAVE*', '*ENTER_LEAVE*' depending on whether the users are interested in trajectories that enter/leave or enter&leave the area within the given time period.

HERMES ARCHITECTURE

HERMES can be utilized in a real world scenario to assist a database developer in modeling, querying and analyzing moving object databases. A straightforward utilization scenario is to design and construct a spatio-temporal object-relational database schema using HERMES and build an application by transacting with this database. Figure 9 illustrates such a scenario on top of *Oracle* ORDBMS. In this case and in order to specify the database schema, the database designer writes scripts in the syntax of the *Data Definition Language* (DDL), which in this case is the PL/SQL, extended with the spatio-temporal operations previously introduced.

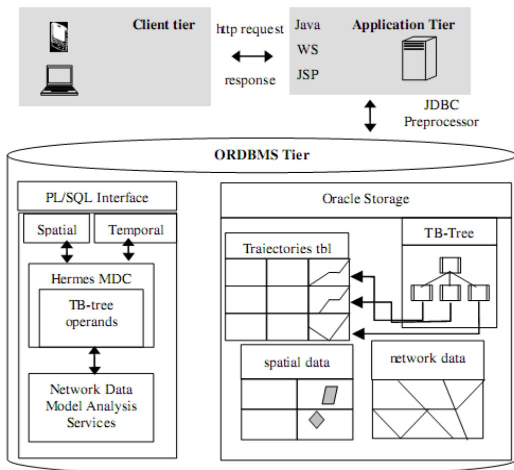

Figure 9 The architecture of the HERMES

To build an application on top of such a database for creating objects, querying data and manipulating information, the application developer writes a source program (for instance) in *Java* (or *JSP* in case of web-based applications) wherein he/she can embed *PL/SQL scripts* that invoke object constructors and methods from HERMES. The *JDBC pre-processor* integrates the power of the programming language with the database functionality offered by the extended PL/SQL and together with the *ORDBMS Runtime Library* generate the application's executable. By writing independent stored procedures that take advantage of HERMES functionality and by compiling them with the *PL/SQL Compiler*, is another way to build a spatio-temporal application. Figure 10 depicts such an application which also acts as a web-based visual query builder for HERMES.

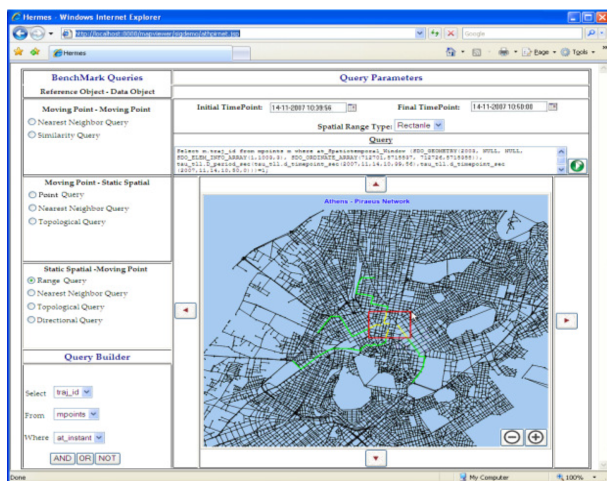

Figure 10 A visual query builder for HERMES

AN APPLICATION OF HERMES TO VEHICLE TRAFFIC ANALYSIS

To demonstrate the functionality of HERMES, in the following paragraphs we present an application example related to vehicle traffic analysis. The motivation is that a courier company, whose vehicles are equipped with GPS devices transmiting their space-time location to a central MOD, needs a flexible way to manage and analyse the motion of the vehicles. One can enumerate a series of benefits to be gained from a possible optimization of the movements of the couriers, such as, personnel's control, better and faster customer services, minimization of operational costs, enhanced decision making. By utilizing this application example, the expressive power and the applicability of HERMES in such a commercial domain are demonstrated. We note that the subsequent discussion and terminology follows the syntax of HERMES as implemented in Oracle ORDBMS. We have already mentioned that the core of HERMES has also been implemented in (Boulahya., 2009) inside another ORDBMS, namely the PostGIS. This actually proves the correctness of the design of HERMES on top of extensible ORDBMS that have OGC-compliant spatial extensions. The differences in the syntax between the two implementations are minor (mainly due to the syntax differences of the two static spatial extensions) (Zimányi, 2010), while we are in the process of testing the compatibility between the results of the operations. In order to present the capabilities of HERMES, we build the following database:

Highways (name: Varchar2, line: SDO_GEOMETRY)

Landmarks (name: Varchar2, kind: Varchar2, location: SDO_GEOMETRY)

Vehicles (id: Varchar2, type: Varchar2, route: Moving_Point)

Highways relation is a set of linestring geometries along which the vehicles are supposed to be moving. Landmarks relation contains locations of certain landmarks, such as petrol stations, etc. Vehicles relation identify the route of a lorry that is modeled as a moving point, while *type* attribute stamps each vehicle with a characteristic description of each kind (e.g. truck, motorbike, etc.). Furthermore, field route of relation Vehicles is indexed by a TB-tree.

In the following paragraphs, we illustrate a composite spatio-temporal scenario (in the form of a series of queries) in the domain of the application example. The linguistic description of each query is followed by the implementation of the query in the

form of a *PL/SQL* block, as well as by an abstract presentation of the way that such a query is resolved. This scenario illustrates the expressive power and the spatio-temporal query capabilities added to *PL/SQL* by HERMES.

(Q1) Find all vehicles moving inside a given region and time period?

PL/SQL block for Q1:

```
DECLARE
   region SDO_GEOMETRY :=
    SDO_GEOMETRY(2003, NULL, NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(489048,4203749,
                                        90032,4205990));
BEGIN
   SELECT
   TB_MP_IN_SPATIOTEMPORAL_WINDOW(
   region, tau_tll.d_period_sec(
     tau_tll.D_Timepoint_Sec(2010,7,9,10,35,0),
     tau_tll.D_Timepoint_Sec(2010,7,9,10,55,0))
   )
   FROM Vehicles;
END;
```

The query Q1 is the classic spatio-temporal range query that is answered with the employment of the TB-tree operators, by simply invoking function *tb_mp_in_spatiotemporal_window*. Actually, this is the query depicted in the query builder in Figure 10.

(Q2) If vehicle 'X' is in the result set of Q1, when and where did it enter the region?

PL/SQL block for Q2:

```
DECLARE
   truckX Moving_Point;
   truckX_IN_region Moving_Point;
   temp_projection
TAU_TLL.TEMP_ELEMENT_SEC;
   when TAU_TLL.TIMEPOINT_SEC;
   where SDO_GEOMETRY;
BEGIN
   SELECT route INTO truckX FROM Vehicles
   WHERE id='X';
   truckX_IN_region:=
          truckX.f_intersection(region);
   temp_projection:=
          truckX_IN_region.f_temp_element();
   when :=
          temp_projection.te(
          temp_projection.te.FIRST).b;
   where := truckX_IN_region.f_initial();
END;
```

To address Q2, we demonstrate how we can restrict a moving point inside a static spatial region and how to temporally and spatially project this restricted moving point in its initial position. The result of such an operation (f_intersection) is another Moving_Point. By temporally projecting it (f_temp_element) on the continuous time line and finding the temporal element that consists of the time periods for which are defined the unit moving objects of the moving courier, we can estimate the timepoint when initially entered the given region. In addition, by applying the f_initial method, we can locate the point that this happened.

(Q3) A variant of Q3 would be to find all trajectories entering a given spatio-temporal range.

PL/SQL block for Q3:

```
    SELECT * FROM TABLE(
      TB_TOPOLOGICAL_QUERY(
        SDO_GEOMETRY(2003, NULL, NULL,
        SDO_ELEM_INFO_ARRAY(1,1003,3),
        SDO_ORDINATE_ARRAY(489048,4203
                                    749, 90032,4205990)),
  tau_tll.d_period_sec(
    tau_tll.D_Timepoint_Sec(2010,7,9,10,35,0),
    tau_tll.D_Timepoint_Sec(2010,7,9,10,55,0)),
                                          'ENTER'))
```

(Q4) What distance has vehicle 'X' travelled inside the region?

PL/SQL block for Q4:

```
DECLARE
   distance double;
BEGIN
   distance:= LENGTH (
   INTERSECTION (region,
                        truckX.f_trajectory()));
END;
```

This query is resolved by finding the intersection of the region with the trajectory followed by the courier (f_trajectory operation). This intersection is a *LineString* geometry that restricts the route of the courier *inside* the region and by applying the LENGTH spatial operator upon the resulted *LineString* we compute the required distance.

(Q5) Give a list of options to the driver of vehicle 'X' to refuel the vehicle within the next 2km

PL/SQL block for Q5:

```
BEGIN
   SELECT name, location FROM Landmarks
   WHERE kind = 'petrol station' AND
   truckX.f_within_distance(2000,location, 0.005,
   TAU_TLL.now()) = 'TRUE';
END;
```

In order to provide the list of petrol stations (Q5), we select landmarks that are petrol stations and the courier is within the specified distance (f_within_distance) from them at the time the query is invoked.

Based on related research work (Pelekis, Theodoulidis, Kopanakis, & Theodoridis, 2004) queries like the above constitute a minimum functionality a MOD system should provide. Furthermore, the usefulness and applicability of the server-side extensions provided by HERMES have been proved in (Pelekis, Theodoridis, Vaosinakis, & Panayiotopoulos, 2006) and (Pelekis, Frentzos, Giatrakos, & Theodoridis, 2008) by developing benchmark queries proposed in (Theodoridis, 2003) for the evaluation of systems supporing Location-Based Services.

BUILDING REAL MOBILITY-CENTRIC APPLICATIONS ON TOP OF HERMES

The best way to evaluate HERMES is to assess the realization of its initial goal, which is to provide a complete framework for developing MOD-related applications. In the previous section we provided a sketch for building a specific application related to vehicle traffic analysis, while in this section we demonstrate this by briefly presenting successfull applications of HERMES in four different domains, namely in *trajectory data warehouses* (i.e. TDW (Marketos et al., 2008)), in *moving object data mining query languages* (i.e. DAEDALUS tool (Ortale et al., 2008), in *semantic enrichment of movement patterns* (i.e. ATHENA tool (Baglioni, Macedo, Renso, Trasarti, & Wachowicz, 2009)), and in *privacy-aware trajectory tracking query engines* (i.e. HERMES++ tool (Gkoulalas-Divanis, & Verykios, 2008)). We would like to note that the above works are a subset of tools and methods developed as a result of a European-wide research project called GeoPKDD – (Geographic Privacy-Aware Knowledge Discovery and Delivery). HERMES is also a prototype outcome of GeoPKDD designed to be the MOD management infrastructure of such tools. Of course, in order to support such diverse applications domains we have designed and incorporated into HERMES several specialized operations (e.g. a trajectory anonymizer operator for (Gkoulalas-Divanis, & Verykios, 2008)).

Trajectory data warehouses – Data Warehouses have shown their importance in real-world applications (Rifaie, Kianmehr, Alhajj, & Ridley, 2009). TDW aim at developing a multi-dimensional model suitable for online analytical processing (OLAP) of trajectory data, such as drill-down and roll-up operations. In order to design a trajectory warehouse architecture, one should first identify the differences from conventional warehouse approaches and then to devise appropriate extensions. There are three steps so as to realize the development of a TDW. At the first step the design of a MOD and of a multidimensional data model (i.e. trajectory data cube) takes place. At the second step, preprocessing (i.e. cleaning, consistency checking) and loading of raw movement data into the MOD occurs, while once trajectories have been stored in the MOD, the Extract-Transform-Load (ETL) phase is executed in order to feed the TDW and the measures of the data cube are calculated. In (Marketos et al., 2008; Leonardi et al., 2010; Raffaeta et al. 2011) HERMES has been employed as the infrastructure to develop the above described process in a huge, real trajectory dataset, where due to the size of the dataset, the existence of efficient, scalable querying processing operators to support ETL was a key requirement.

Moving object data mining query languages (MO-DMQL) – In (Ortale et al., 2008) the authors proposed DAEDALUS, a formal framework and system, that defines knowledge discovery processes as a progressive combination of mining and querying operators. The heart of DAEDALUS is the MO-DMQL query language that extends SQL in two aspects, namely a pattern definition operator and functionality to uniformly manipulate both raw trajectory data and unveiled movement patterns. DAEDALUS system has been implemented as a query execution layer on top of the HERMES. More specifically, the role of HERMES in DAEDALUS is two-fold; to act as a repository for movement data and secondly to give the basic building block that allows defining models' representation and storage.

Semantic enrichment of movement patterns – Having as aim to provide a model for the conceptual representation and deductive reasoning of trajectory patterns obtained from mining raw trajectories, the authors in (Baglioni et al., 2009) have developed ATHENA tool, which employs ontologies for the semantic enrichment of trajectories. This is achieved by means of a semantic enrichment process, where raw trajectories are enhanced with semantic information and integrated with geographical knowledge encoded in an ontology. To highlight this process imagine that a user poses a query using the ontology concepts where trajectories/patterns are classified by a reasoner. The ontology is populated by instances coming from a MOD storing semantic trajectories, patterns and auxiliary geographical features. Again, HERMES supports all the spatio-temporal data management requirements raised by ATHENA. The overall undertaking was evaluated in a real-world case study posing as analysis objective the understanding of tourist movements in Milan's metropolitan area.

Privacy-aware trajectory tracking query engines – Due to the very nature of movement data, lately a new line of research has emerged that investigates

safeguards to enforce so as to ensure the privacy of the individuals, whose movement is recorded. HERMES++ (Gkoulalas-Divanis, & Verykios, 2008) which has been designed on top of HERMES describes such a privacy aware trajectory tracking query engine, where subscribed users can gain restricted access to an in-house trajectory data warehouse, to perform certain analysis tasks. In addition to regular queries involving non-spatial non-temporal attributes, the engine supports a variety of spatiotemporal queries, including range queries, nearest neighbor queries and queries for aggregate statistics. The query results are augmented with fake trajectory data (dummies) to fulfill the requirements of K-anonymity.

CONCLUSIONS AND OUTLOOK

In this paper, a data management framework for TD, called HERMES, was introduced. This framework is a system extension that provides spatio-temporal functionality to OGC-compatible ORDBMS and supports mobility-centric applications. Future work includes inclusion of query optimization strategies using the extensibility interfaces of current ORDBMS in order to enhance the performance of HERMES. Furthermore, we plan to extend HERMES with data mining query operators aiming at transforming the formulated language to a data mining query language for TD.

ACKNOWLEDGMENT

Research partially supported by the FP7 ICT/FET Project MODAP (Mobility, Data Mining, and Privacy) funded by the European Union (URL: www.modap.org). Elias Frentzos is supported by the Greek State Scholarships Foundation.

REFERENCES

Baglioni, M., Macedo, J.A.F., Renso, C., Trasarti, R., & Wachowicz, M. (2009). Towards semantic interpretation of movement behavior, In *Proceedings of the 12th AGILE International Conference on Geographic Information Science*.

Boulahya., S. (2009). Représentation et interrogation de données spatio-temporelles: Cas d'étude sur postgresql/postgis. *Masters' Thesis, Department of Computer and Decision Engineering, Université Libre de Bruxelles*, Brussels, Belgium, (in French).

Chakka, V.P., Everspaugh, A., & Patel., J. (2003). Indexing large trajectory data sets with SETI. In *Proceedings of CIDR*.

Chatterjee, K., & Chen, S.C. (2010). HAH-tree: towards a multidimensional index structure supporting different video modeling approaches in a video database management system. *International Journal of Information and Decision Sciences* (IJIDS), *2*(2), 188-207.

Forlizzi, L., Güting, R. H., Nardelli, E., & Schneider, M. (2000). A data model and data structures for moving objects databases. In *Proceedings of the ACM SIGMOD Int'l Conf. on Management of Data*.

Frentzos, E. (2008). Trajectory data management. *PhD Thesis, Department of Informatics, University of Piraeus*.

Frentzos, E., Gratsias, K., Pelekis, N., & Theodoridis Y. (2007). Algorithms for nearest neighbor search on moving object trajectories. *Geoinformatica, 11*(2), 159-193.

GeoPKDD (Geographic Privacy-aware Knowledge Discovery and Delivery) FP6-14915 IST/FET Project, funded by the European Commission. URL: www.geopkdd.eu.

Gkoulalas-Divanis, A., & Verykios, V. S. (2008, 07). A privacy aware trajectory tracking query engine. *ACM SIGKDD Explorations*, *10*(1), 40-49.

Güting, R.H., Bohlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N.A., Schneider, M., & Vazirgiannis, M. (2000). A foundation for representing and querying moving objects. *ACM Transactions on Database Systems*, *25*(1), 1-42.

Kargin, B., Basoglu, N., & Daim, T. (2009). Adoption factors of mobile services. *International Journal of Information Systems in the Service Sector* (IJISSS), *1* (1): 15-34.

Lema, J.A.C., Forlizzi, L., Güting, R.H., Nardelli, E., & Schneider, M. (2003). Algorithms for moving objects databases. *The Computer Journal 46*(6), 680-712.

Leonardi, L. Marketos, G, Frentzos, E., Giatrakos, N., Orlando, S., Pelekis, N., Raffaetà, A., Roncato, A., Silvestri, C., & Theodoridis, Y. (2010). T-warehouse: visual olap analysis on trajectory data. In *Proceedings of the 26th IEEE International Conference on Data Engineering*.

Marketos, G, Frentzos, E., Ntoutsi, I., Pelekis, N., Raffaeta, A., & Theodoridis, Y. (2008). Building real-world trajectory warehouses". In *Proceedings of the 7th International ACM SIGMOD Workshop on Data Engineering for Wireless and Mobile Access*.

Ni, Y., & Ravishankar, C. (2007). Indexing spatio-temporal trajectories with efficient polynomial approximations, *IEEE Transactions on Knowledge Discovery*, *19*(5), 663-678.

Ortale, R., Ritacco, E., Pelekis, N., Trasarti, R., Costa, G., Giannotti, F., Manco, G., Renso, C., & Theodoridis, Y. (2008). The Daedalus framework: progressive querying and mining of movement data. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*.

Pelekis, N. (2002). STAU: A spatio-temporal extension to oracle dbms. *PhD Thesis, UMIST, Department of Computation*.

Pelekis, N., Frentzos, E., Giatrakos, N., & Theodoridis, Y. (2008). Hermes: Aggregative lbs via a trajectory db engine. In *Proceedings of the ACM SIGMOD Conference*.

Pelekis, N. Kopanakis, I., Ntoutsi, I., Marketos, G., Andrienko, G., & Theodoridis, Y. (2007). Similarity search in trajectory databases. In *Proceedings of the 14th IEEE International Symposium on Temporal Representation and Reasoning* (TIME 2007).

Pelekis, N., & Theodoridis, Y. (2010, 07). An oracle data cartridge for moving objects. *Information Systems Laboratory, Department of Informatics, University of Piraeus, UNIPI-ISL-TR-2010-01*.
http://isl.cs.unipi.gr/publications.html.

Pelekis, N., Theodoulidis, B., Kopanakis, I., & Theodoridis, Y. (2004,06). Literature review of spatio-temporal database models. *Knowledge Engineering Review*, *19*(3), 235-274.

Pelekis, N., Theodoridis, Y., Vosinakis, S., & Panayiotopoulos, T. (2006). Hermes – A framework for location-based data management. In *Proceedings of the 10th Int'l Conference on Extending Database Technology*.

Pfoser, D., Jensen, C.S., & Theodoridis Y. (2000). Novel approaches to the indexing of moving object trajectories. In *Proceedings of the International Conference on Very Large Databases*.

Raffaetà, A., Leonardi, L., Marketos, G., Andrieko, G., Andrienko, N., Frentzos, E., Giatrakos, N., Orlando, S., Pelekis, N., Roncato, A., & Silvestri C. (2011). Visual mobility analysis using T-warehouse. *International Journal of Data Warehousing & Mining*, to appear.

Rifaie, M., Kianmehr, K. , Alhajj, R. , & Ridley M. J. (2009). Data modelling for effective data warehouse architecture and design. *International Journal of Information and Decision Sciences* (IJIDS), *1*(3), 282-300.

Theodoridis, Y. (2003). Ten benchmark database queries for location-based services, *The Computer Journal*, *46*(6), 713-725.

Theodoridis, Y., Vazirgiannis, M., & Sellis, T. (1996). Spatio-temporal indexing for large multimedia applications. In *Proceedings of ICMCS*.

Zhang, P. (2003). The spatial movement extensions of stau. *MPhil Thesis, UMIST, Department of Computation*.

Zimányi, E. (2010). personal communication