On the Support of Mobility in ORDBMS

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

On the Support of Mobility in ORDBMS

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

Composition of space and mobility in a unified data framework results into Moving Object Databases (MOD). MOD management systems support storage and query processing of non-static spatial objects and provide essential operations for higher level analysis of movement data. The goal of this paper is to present HERMES MOD engine that supports the aforementioned functionality through appropriate data types and methods in Object-Relational DBMS (ORDBMS) environments. In particular, HERMES exploits on the extensibility interface of ORDBMS that already have extensions for static spatial data types and methods that follow the Open Geospatial Consortium (OGC) standard, and extends the ORDBMS by supporting time-varying geometries that change their position and/or extent in space and time dimensions, either discretely or continuously. It further extends the data definition and manipulation language of the ORDBMS with spatio-temporal semantics and functionality.

Keywords: HERMES, Moving Object Databases, Spatio-temporal Data Management, ORDBMS extension, OGC spatial types.

INTRODUCTION

Due to the explosion of mobile devices, the positioning technologies and the low data storage cost, one of the most important assets of knowledge intensive organizations working with movement data, traffic engineering, climatology, social anthropology and zoology, studying vehicle position data, hurricane track data, human and animal movement data, respectively etc.) is the data itself. Spatial database research has focused on supporting the modeling and querying of geometries associated with objects in a database (Güting, 1994). Regarding static spatial data, the major commercial as well as open source database management systems (e.g., DB2, MySQL, Oracle, Postgis, SQL Server) already provide appropriate data management and querying mechanisms that conform to Open Geospatial Consortium (OGC) standards (Open geospatial consortium, 2010). On the other hand, temporal databases have focused on extending the knowledge kept in a database 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*) (Tansel et al., 1993). About a decade's effort attempts to achieve an appropriate kind of interaction between both subareas of database research. Spatio-temporal databases are the outcome of the aggregation of time and space into a single framework (Koubarakis, & Sellis, 2003) with up-to-date reviews of spatio-temporal models and systems proposed in the literature found in (Pelekis, Theodoulidis, Kopanakis, & Theodoridis, 2004) and (Frentzos, Pelekis, Ntoutsi, & Theodoridis, 2008), respectively. As delineated in these papers, 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. For the previous reasons, the field of the MOD has emerged (Güting, 2000), and has been shown (Pelekis et al., 2004) that it presents the most desirable properties among the proposals. However, although a lot of research has been carried out in the field of MOD, the efforts are independent trying to deal with specific problems and do not pay attention into embedding the proposed solutions (i.e. query processing algorithms) on top of existing DBMS where real world organizations base on. Towards this direction, the pioneer work of (Güting et al., 2000; Forlizzi, Güting, Nardelli, & Schneider, 2000; Lema, Forlizzi, Güting, Nardelli, & Schneider, 2003) have proposed the SECONDO system (Almeida, Güting, & Behr, 2006). However, SECONDO in contradiction to our approach is a stand-alone system, built from scratch, its design does not utilize the provided spatial extensions of existing ORDBMS, it does not conform to the OGC standards as it does not follow any predefined data model (Dieker, & Güting, 2000) and as such it is not embeddable into the DBMS infrastructure of an organization, where pure static spatial, as well as other types of data is stored.

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 MOD 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 or extent 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 exactly the challenge adopted by this paper. In detail, we present an integrated and comprehensive design of moving object data types in the form of extensible modules that can be embedded in OGC-compliant Object-Relational Database Management Systems (ORDBMS) taking advantage of their extensibility interface. The proposed HERMES *MOD Engine* provides the functionality to construct a set of moving, expanding and/or shrinking geometries, which are just variables of simple continuous functions that obtain hypostasis when projected to the spatial domain (i.e. becoming OGC spatial data types) at a specific instance in time. Each one of these moving objects is supplied with a set of methods that facilitate the user to query and analyze spatio-temporal data. Embedding this functionality offered by HERMES in an ORDBMS data manipulation language, one obtains a flexible, expressive and easy to use query language for moving objects that was not available so far in real OGC-compliant ORDBMS.

The implementation of such a framework is based on a set of basic types including base data types (i.e. integer, real, string and boolean, available in all DBMS), together with spatial data types offered by spatial extensions of OGC-compliant ORDBMS and temporal data types introduced in a temporal extension, called *TAU Temporal Literal Library* (TAU-TLL) (Pelekis, 2002). Based on these temporal and spatial object data types and the ideas behind the abstract data types for moving objects that have been introduced in (Güting et al., 2000), this paper discusses the design principles and the implementation issues concerning HERMES. The values of such moving types are functions that associate each instant in time with an OGC spatial type, in contradiction to (Güting et al., 2000) whose design does not follow the OGC standards. A rich palette of suitable operations is defined on these types to support querying and to make moving object data management easier, more natural and sensible to users and applications.

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

- We present a datatype-oriented model and an extension of SQL-like query language for supporting the modeling and querying of MOD on top of OGC-compliant ORDBMS.
- We describe the physical representation design decisions and the architectural aspects of our server-side MOD database engine, as well as the formulated interface (in terms of operators registered in the ORDBMS) for building advanced mobility-related applications.

To the best of our knowledge, HERMES is the first work that provides a complete framework for building MOD applications, 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 moving objects introduced in HERMES in an abstract way and then, we discuss implementation aspects. An appropriate set of operations that extend the data definition and manipulation language of an ORDBMS with spatio-temporal semantics is subsequently discussed. An extensive discussion on the comparison of HERMES functionality with related work appears follows. Finally, we conclude the paper, also pointing out some interesting future research directions.

A DATA TYPE SYSTEM FOR MOVING OBJECTS

The basic modeling primitives of the proposed moving object data type system are objects and literals. An *object* is a computational entity with a unique object identifier that encapsulates both state and behavior. The *state* of an object is defined by the values it carries for a set of properties. These *properties* can be attributes of the object itself or relationships between the object and one or more other objects. The *behavior* of an object is defined by a set of operations that can be executed on or by the object. On the other hand, a *literal* is a computational entity that has only state. Let V be a universe of all possible computational entities, containing objects and literals. A *type* is a set of elements of V that obey some technical properties. Each type is associated with a predicate function defined over the V. A value $v \in V$ satisfies a type iff the predicate is true for that value. A value that satisfies a type is called *member* of the type. A *type system* is a collection of types.

Types in the so-called *MOD Type System* are divided into *Base Types BT*, pure *Temporal Types TT*, pure OGC-compliant *Spatial Types ST* and *Moving Types MT*, i.e., the proposed *MOD Type System* is defined as:

$$MOD = BT \cup TT \cup ST \cup MT \quad (1)$$

Figure 1 illustrates, in UML notation, all types in *MOD Type System*.

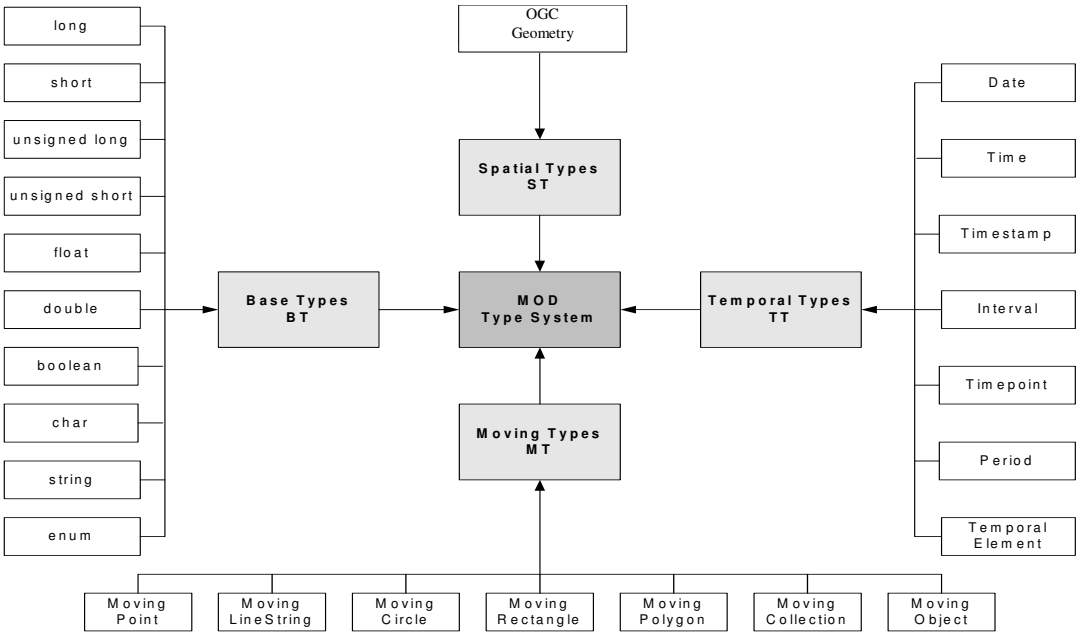

Figure 1 MOD Type System

Base, Temporal and Spatial Types

Base types are the standard database types built into any DBMS, such as integer and real (float) numbers, alphanumeric strings and booleans. These types form a subset of the Atomic Literal Types needed to define temporal types. The set *ALT* of *Atomic Literal Types* is defined as:

$$\begin{aligned} ALT = {} & \Pi\, boolean\, T \cup \Pi\, char\, T \cup \Pi\, short\, T \cup \\ & \Pi\, ushort\, T \cup \Pi\, long\, T \cup \Pi\, ulong\, T \cup \\ & \Pi\, float\, T \cup \Pi\, double\, T \cup \Pi\, octet\, T \cup \\ & \Pi\, string\, T \cup \Pi\, enum\, T \end{aligned} \quad (2)$$

where $\Pi\, *\, T$ denotes the domain of type *. For example, $\Pi\, boolean\, T = \{true, false\}$, $\Pi\, char\, T = \{x \mid x \in ASCII\}$, and so on.

Moving from base to temporal types, the set *TLT* of Temporal Literal Types is defined as (Kakoudakis, 1996; Pelekis, 2002):

$$\begin{aligned} TLT = {} & \Pi\, date\, T \cup \Pi\, time\, T \cup \Pi\, timestamp\, T \cup \\ & \Pi\, interval\, T \cup \Pi\, timepoint\langle g \rangle\, T \cup \\ & \Pi\, period\langle g \rangle\, T \cup \\ & \Pi\, temporalElement\langle g \rangle\, T \end{aligned} \quad (3)$$

Basically, TLT augments the four temporal literal data types found in *ODMG* object model (Cattel, & Barry, 1997) (namely, *Date*, *Time*, *Timestamp* and *Interval*) with three new temporal object data types (namely, *Timepoint*, *Period* and *Temporal Element*). The widely used *Gregorian calendar* is implemented and the *discrete* model of time is adopted, where time is isomorphic to the integers because of its better representation and manipulation on databases. Time axis is partitioned into a finite number of discrete segments, called *granules*. The choice of a partitioning scheme is termed as *granularity*. The granularity of the timestamp that a fact is associated with denotes the precision to which the timestamp can be represented. Time order refers to whether the time axis can be always considered as linear or non-linear. In the linear model, time advances from past to future in a totally ordered form. The non-linearity of the time axis deals with aspects of the time such as *periodic time* and *branching time* (Theodoulidis , & Loucopoulos, 1991). Formally:

$$\begin{aligned} & date =_d \langle\!\langle year: GrYear, month: GrMonth, day: GrDay \rangle\!\rangle \\ & time =_d \langle\!\langle hour: GrHour, minute: GrMinute, second: GrSecond \rangle\!\rangle \\ & timestamp =_d date \,||\, time \\ & interval =_d \langle\!\langle day: long, hour: GrHour, minute: GrMinute, second: GrSecond \rangle\!\rangle \\ & timepoint\langle g \rangle =_d tp\langle g \rangle \cup STV \\ & period\langle g \rangle =_d \{\langle\!\langle start:Timepoint\langle g \rangle, end:Timepoint\langle g \rangle \rangle\!\rangle \mid start \le end\}, g \in granularity \\ & temporalElement\langle g \rangle =_d \{te: set\langle period\langle g \rangle \rangle \mid \forall i, j \cdot i \neq j \Rightarrow te_i \cap te_j = \varnothing\} \end{aligned} \quad (4)$$

where the set *granularity* that contains elements that represent time accuracy according to the time divisions in the *Gregorian* calendar:

granularity $= \{YEAR, MONTH, DAY, HOUR, MINUTE, SECOND\}$, $tp\langle year\rangle =_d \langle\langle year: GrYear\rangle\rangle$, $tp\langle month\rangle =_d tp\langle year\rangle \parallel \langle\langle month: GrMonth\rangle\rangle$, ..., $tp\langle second\rangle =_d tp\langle minute\rangle \parallel \langle\langle second: GrSecond\rangle\rangle$ and $STV =_d \{beginning, forever, now\}$.

The four temporal literal data types found in *ODMG* object model (Cattel et al., 1997) are augmented with three new temporal object data types presented below:

- *Timepoint*: extends the *Timestamp* data type to include granularity. The new data type is a subtype of the *Timestamp* data type. It inherits all the properties and the operations that are defined for the *Timestamp* data type. It refines all the operations, which had as argument *Timestamp* to *Timepoint*. *Beginning* and *forever* are defined to be members of *timepoint* such as $\forall\, t \in timepoint\langle g\rangle \cdot beginning \le t \le forever$.
- *Period*: is used to represent an anchored duration of time, that is, duration of time with starting and ending points. A period has an associated granularity. The period is the composition of two timepoints with the constraint that the timepoint that starts the period equals or precedes the timepoint that terminates it. It is assumed that both timepoints have the same granularity. There are four categories of periods depending on whether they include their starting and/or their ending timepoints or not: $[t_1, t_2]$ (closed-closed), $[t_1, t_2)$ (closed-open), $(t_1, t_2]$ (open-closed), and (t_1, t_2) (open-open). *TAU Model* supports only closed-open periods, with which it is possible to model any other category. For example, the period $[t_1, t_2]$ is equivalent to the period $[t_1, t_2+1$ "granule"). The meaning of "1 granule" depends on the granularity of the period. For instance, if the granularity is day then the period $[t_1, t_2]$ is equivalent to the period $[t_1, t_2+1*DAY)$.
- *Temporal Element*: is used to represent a finite union of disjoint periods. Temporal elements are closed under the set theoretic operations of union, intersection and complementation.

On the other hand, spatial types (point, line segment, rectangle, etc.) are supported by another component of the *MOD* type system architecture, called *OGC Geometry*. Such a spatial extension is found in several state-of-the-art ORDBMS 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 *MOD* type system we propose in the sequel.

Abstract Definitions of Moving Object Data Types

In this work, we adopt and extend the *sliced representation* concept (Güting et al., 2000; Forlizzi et al., 2000; Lema et al., 2003) and utilize it in the implementation of the MOD type system that results to HERMES. In order to use the sliced representation to define a moving type, one has to decompose the definition of each moving type into several definitions, one for each of the slices that corresponds to a simple function (i.e. corresponding to a so-called *Unit_Function* type), which is valid for a period of time, 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*.

The *Unit_Function* object type (Pelekis, & Theodoridis, 2006) is 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. Consequently, in the general case the *Unit_Function* is defined as follows:

$$Unit\_Function =_d \langle\langle\, x_i{:}double,\ y_i{:}double,\ x_e{:}double,\ y_e{:}double,\ x_c{:}double,\ y_c{:}double,\ v{:}double,\ a{:}double,\ flag{:}TypeOfFunction \rangle\rangle \quad (5)$$

where $TypeOfFunction = \{PLNML\_1, ARC\_\langle 1..8\rangle, CONST\}$, meaning 1st order polynomial, one of the eight possible circular arcs, and constant function, respectively.

Combining *time period* and *simple function* 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. Formally:

$$Unit\_Moving\_Point =_d \langle\langle p{:}\ period\langle SECOND\rangle,\ m{:}\ Unit\_Function \rangle\rangle \quad (6)$$

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 circle and rectangle geometry constructs that change

their position and/or extent over time. Formally:

$$Unit\_Moving\_Rectangle =_d \{\langle ll: Unit\_Moving\_Point, ur: Unit\_Moving\_Point\rangle \mid equal(ll.p, ur.p)\} \quad (7)$$

$$Unit\_Moving\_Circle =_d \{\langle f: Unit\_Moving\_Point, s: Unit\_Moving\_Point, t: Unit\_Moving\_Point\rangle \mid equal(f.p, s.p, t.p)\} \quad (8)$$

For modeling the subsequent object types (*Unit_Moving_Polygon, Unit_Moving_LineString*) an intermediate object type that represents the simplest built-in constituent of these types is needed. This requirement is met by the *Unit_Moving_Segment* object, which models a simple line or arc segment that changes its shape and size according to its starting and ending unit moving points. This is clarified in Figure 2 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 moving point mp_1 follows a simple linear trajectory, while the ending moving point mp_2 follows an arc trajectory.

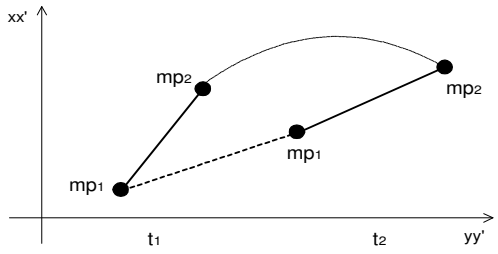

Figure 2 Linear Unit_Moving_Segment with its first Unit_Moving_Point mp_1 moving linearly and the second mp_2 moving on a circular arc

Formally:

$$Unit\_Moving\_Segment =_d \{\langle b: Unit\_Moving\_Point, e: Unit\_Moving\_Point, m: Unit\_Moving\_Point, kind: TypeOfSegment\rangle \mid (kind = SEG \Rightarrow equal(b.p, e.p)) \wedge (kind = ARC \Rightarrow equal(b.p, e.p, m.p))\} \quad (9)$$

$$Unit\_Moving\_Linestring =_d \{l: set\langle Unit\_Moving\_Segment\rangle \mid \forall i, j \in ulong: i \neq j \Rightarrow equal(l_i.b.p, l_j.e.p)\} \quad (10)$$

$$Unit\_Moving\_Polygon =_d \{\langle l: set\langle Unit\_Moving\_Segment\rangle, hole: boolean\rangle \mid \forall i, j \in ulong: i \neq j \Rightarrow equal(l_i.b.p, l_j.e.p)\} \quad (11)$$

where $\Pi\, TypeOfSegment\ T = \{SEG, ARC\}$ and SEG, ARC denote the two alternative modes of interpolation in between two end points (line segment vs. arc, respectively).

Having defined the fundamental unit moving types, we now introduce the moving types that play the dominant role in our spatio-temporal data type system. The process that we followed to define the moving types is to introduce a moving type as a collection of the corresponding unit moving type, which means, in terms of object orientation, that there exists a composition relationship between the unit moving type and the moving type. As such, the *Moving_Point*, *Moving_Circle*, *Moving_Rectangle*, *Moving_LineString* and *Moving_Polygon* object types are introduced as a collection of *Unit_Moving_Point, Unit_Moving_Circle, Unit_Moving_Rectangle, Unit_Moving_LineString, Unit_Moving_Polygon* object types, respectively. Formally:

$$Moving\_Point =_d \{p: set\langle Unit\_Moving\_Point\rangle \mid \forall i, j \in ulong, 1 \leq i, j \leq |set\langle Unit\_Moving\_Point\rangle| : j = i+1 \Rightarrow p_i.p < p_j.p \wedge \neg overlaps(p_i.p, p_j.p) \wedge \forall t \in double: inside(t, p_i.p) \Rightarrow at\_instant(p, t) \in OGC\text{-}GEOMETRY_{GTYPE} = point\} \quad (12)$$

$$Moving\_Rectangle =_d \{r: set\langle Unit\_Moving\_Rectangle\rangle \mid \forall i, j \in ulong, 1 \leq i, j \leq |set\langle Unit\_Moving\_Rectangle\rangle| : j = i+1 \Rightarrow r_i.ll.p < r_j.ur.p \wedge \neg overlaps(r_i.ll.p, r_j.ur.p) \wedge \forall t \in double: inside(t, r_i.ll.p) \Rightarrow at\_instant(r, t) \in OGC\text{-}GEOMETRY_{GTYPE} = rectangle\} \quad (13)$$

$$Moving\_Circle =_d \{c: set\langle Unit\_Moving\_Circle\rangle \mid \forall i, j \in ulong, 1 \leq i, j \leq |set\langle Unit\_Moving\_Circle\rangle| : j = i+1 \Rightarrow c_i.f.p < c_j.s.p \wedge \neg overlaps(c_i.f.p, c_j.s.p) \wedge \forall t \in double: inside(t, c_i.f.p) \Rightarrow at\_instant(c, t) \in OGC\text{-}GEOMETRY_{GTYPE} = circle\} \quad (14)$$

$$Moving\_LineString =_d \{line: set\langle Unit\_Moving\_LineString\rangle \mid \forall i, j \in ulong, 1 \leq i, j \leq |set\langle Unit\_Moving\_LineString\rangle| : j = i+1 \Rightarrow line_i.l_1.b.p < line_j.l_1.e.p \wedge \neg overlaps(line_i.l_1.b.p, line_j.l_1.e.p) \wedge \forall t \in double: inside(t, line_i.l_1.b.p) \Rightarrow at\_instant(line, t) \in OGC\text{-}GEOMETRY_{GTYPE} = linestring\} \quad (15)$$

$$\begin{aligned}
&Moving_Polygon =_d \{\, pol: \\
&\quad set\langle Unit_Moving_Polygon\rangle \mid \forall i, j \in \\
&\quad ulong, 1 \le i, j \le \mid \\
&\quad set\langle Unit_Moving_Polygon\rangle \mid : j = i+1 \Rightarrow \\
&\quad pol_i.l_1.b.p < pol_j.l_1.e.p \wedge \\
&\quad \neg overlaps(pol_i.l_1.b.p, pol_j.l_1.e.p) \wedge \forall t \in \\
&\quad double: inside(t, pol_i.l_1.b.p) \Rightarrow \\
&\quad at_instant(pol, t) \in OGC\text{-}\\
&\quad GEOMETRY_{GTYPE}=polygon \,\}
\end{aligned} \tag{16}$$

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*. Formally (and assuming that the spatial extension of the underlying ORDBMS supports *multi*-spatial types):

$$\begin{aligned}
&Multi_Moving_Point =_d \{\, multi_mpoint: set\langle \\
&\quad Moving_Point\rangle \mid \forall i, j \in ulong \wedge \forall t \in \\
&\quad double: inside(t, multi_mpoint_i.p_j.p) \Rightarrow \\
&\quad \cup_t (at_instant(multi_mpoint_i, t)) \in OGC\text{-}\\
&\quad GEOMETRY_{GTYPE}=multi\text{-}point \,\}
\end{aligned} \tag{17}$$

$$\begin{aligned}
&Multi_Moving_LineString =_d \{\, multi_mline: \\
&\quad set\langle Moving_LineString\rangle \mid \forall i, j \in ulong \\
&\quad \wedge \forall t \in double: inside(t, \\
&\quad multi_mline_i.line_j.l_1.b.p) \Rightarrow \\
&\quad \cup_t (at_instant(mult_i_mline_i, t)) \in OGC\text{-}\\
&\quad GEOMETRY_{GTYPE}=multi\text{-}linestring \,\}
\end{aligned} \tag{18}$$

$$\begin{aligned}
&Multi_Moving_Circle =_d \{\, multi_mcircle: set\langle \\
&\quad Moving_Circle\rangle \mid \forall i, j \in ulong \wedge \forall t \in \\
&\quad double: inside(t, multi_mcircle_i.c_j.f.p) \Rightarrow \\
&\quad \cup_t (at_instant(multi_mcircle_i, t)) \in OGC\text{-}\\
&\quad GEOMETRY_{GTYPE}=multi\text{-}polygon \,\}
\end{aligned} \tag{19}$$

$$\begin{aligned}
&Multi_Moving_Rectangle =_d \{ \\
&\quad multi_mrectangle: \\
&\quad set\langle Moving_Rectangle\rangle \mid \forall i, j \in ulong \wedge \\
&\quad \forall t \in double: inside(t, \\
&\quad multi_mrectangle_i.r_j.ll.p) \Rightarrow \cup_t \\
&\quad (at_instant(multi_mrectangle_i, t)) \in \\
&\quad OGC\text{-}GEOMETRY_{GTYPE}= \\
&\quad multi\text{-}polygon \,\}
\end{aligned} \tag{20}$$

$$\begin{aligned}
&Multi_Moving_Polygon =_d \{ \\
&\quad multi_mpolygon: set\langle Moving_Polygon\rangle \\
&\quad \mid \forall i, j \in ulong \wedge \forall t \in double: inside(t, \\
&\quad multi_mpolygon_i.pol_j.l_1.b.p) \Rightarrow \\
&\quad \cup_t (at_instant(multi_mpolygon_i, t)) \in \\
&\quad OGC\text{-}GEOMETRY_{GTYPE}= \\
&\quad multi\text{-}polygon \,\}
\end{aligned} \tag{21}$$

An interesting issue here is that the previously mentioned multi-moving types do not carry their own methods interface. All 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, the moving-collection type is able to represent heterogeneous collections of moving types, i.e., collections of different time-varying spatial geometries. Formally:

$$\begin{aligned}
&Moving_Collection =_d \{\, \langle\!\langle\, multi_mpoint: \\
&\quad Multi_Moving_Point, multi_mline: \\
&\quad Multi_Moving_LineString, \\
&\quad multi_mcircle: Multi_Moving_Circle, \\
&\quad multi_mrectangle: \\
&\quad Multi_Moving_Rectangle, \\
&\quad multi_mpolygon: \\
&\quad Multi_Moving_Polygon \,\rangle\!\rangle \mid \\
&\quad \forall i, j \in ulong \wedge \forall t \in double: \\
&\quad inside(t, multi_mpoint_i.p_j.p) \wedge \\
&\quad inside(t, multi_mline_i.line_j.l_1.b.p) \wedge \\
&\quad inside(t, multi_mcircle_i.c_j.f.p) \wedge \\
&\quad inside(t, multi_mrectangle_i.r_j.ll.p) \wedge \\
&\quad inside(t, multi_mpolygon_i.pol_j.l_1.b.p) \Rightarrow \\
&\quad [\, (\cup_t (at_instant(multi_mpoint_i, t))) \cup \\
&\quad (\cup_t (at_instant(multi_mline_i, t))) \cup \\
&\quad (\cup_t (at_instant(multi_mcircle_i, t))) \cup \\
&\quad (\cup_t (at_instant(multi_mrectangle_i, t))) \\
&\quad \cup (\cup_t (at_instant(multi_mpolygon_i, t))) \,] \\
&\quad \in OGC\text{-}GEOMETRY_{GTYPE}=collection \,\}
\end{aligned} \tag{22}$$

Furthermore, the *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 (static) polygon geometry is obviously another *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. In other words, if there is an operation that requests the perimeter of *Moving_Polygon*, then obviously the result of this method is a time-varying real number (*Moving_Real*). Such collapsed moving types like *Moving_Real*, *Moving_String*, and *Moving_Boolean* do not formally exist in our type system but are modeled using the *Moving_Object* type.

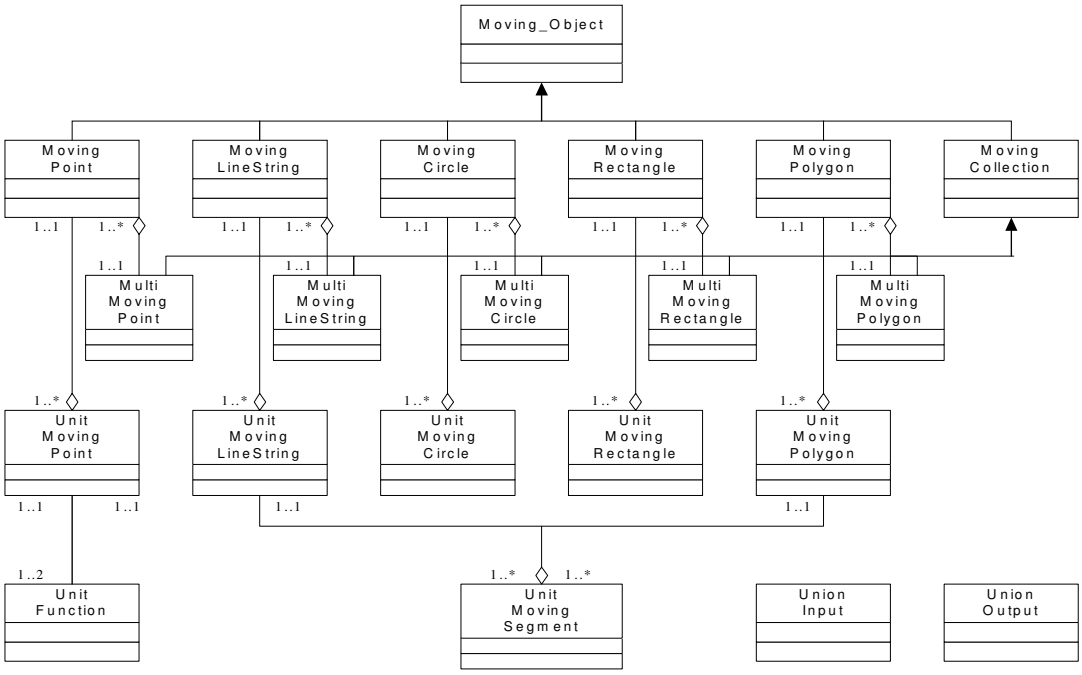

Figure 3 The moving types of MOD type system

Formally:

$$\begin{aligned} Moving\_Object =_d \{\langle\!\langle\ & mobject: Moving\_Object, mpoint: Moving\_Point, \\ & mline: Moving\_LineString, mcircle: Moving\_Circle, mrectangle: Moving\_Rectangle, mpolygon: Moving\_Polygon, mcolection: Moving\_Collection, geometry: GEOMETRY, gtype: GeometryType, \\ & optype: string, arg1: ushort, arg2: ushort, \\ & input: Union\_Input \rangle\!\rangle\} \end{aligned} \tag{23}$$

where *gtype* is a flag that makes *Moving_Object* behave as if it were a simple moving type, Π *GeometryType T* = { *MOBJECT*, *MPOINT*, *MLINE*, *MCIRCLE*, *MRECTANGLE*, *MPOLYGON*, *MCOLLECTION* } and $Union\_Input =_d \langle\!\langle mask: string, tolerance: double, distance: double \rangle\!\rangle$.

Summarizing,

Figure 3 illustrates a UML class diagram for the moving types supported in the proposed *MOD* type system. The following sections describe the design decisions and the implementation details for mapping the *MOD* type system into extensible ORDBMS, as well as essential functionality for extending SQL-like query languages with *MOD* querying constructs.

PHYSICAL MAPPING OF THE HERMES MOD 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 *MOD* types (abstractly described in the previous Section) are mapped to physical structures for storing continuously and discretely time-evolving geometric data into an ORDBMS with OGC-compliant spatial extension. The following subsections propose low-level constructs for the implementation of such objects and illustrate the design decisions and implementation issues considered during development.

Unit Function

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 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 type that changes discretely for a period of time by setting all *Unit_Function* objects of the corresponding unit-moving type 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 geometry, which is valid for the entire period. If at least one of these unit functions is not constant

then the moving type change is continuous for this period of time. In case of a *Moving_LineString* and in order to model a discrete change for a period, the above assignment should take place for each *Unit_Moving_Point* that composes the corresponding *Unit_Moving_LineString*. If this process were continued to all unit-moving types the result would be a completely discretely changing moving geometry.

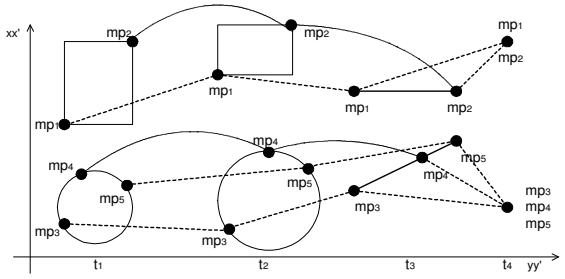

Figure 4 Instances of *Moving_Circle* and *Moving_Rectangle* type objects (and degenerated cases)

Moving_Point*, *Moving_Circle* and *Moving_Rectangle

We construct *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.

Similarly to the *Moving_Point* object, *Moving_Circle* and *Moving_Rectangle* object types are constructed as pointers to collections of *Unit_Moving_Circle* and *Unit_Moving_Rectangle*, respectively.

Let us now examine the structure of *Unit_Moving_Circle* and *Unit_Moving_Rectangle* objects. *Unit_Moving_Circle* consists of three *Unit_Moving_Point* objects, representing the three points 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 and upper-right point needed to define a valid rectangle. Figure 4 illustrates a moving circle and a moving rectangle instantiated at four different time points t_1, t_2, t_3, and t_4, respectively. At time point t_2, it is clear to see the effect of the different interpolation functions and how they affect the position and extent of the mapped geometries, in contrast to time point t_1. At time point t_3, a degenerated moving circle and a degenerated moving rectangle are presented, meaning that the three unit moving points that compose the moving circle become co-linear and the two unit moving points that compose the moving rectangle form a line segment that is parallel to either *xx'* or *yy'* axis. At timepoint t_4, another collapsed state is depicted, where all unit-moving points become equal. HERMES implementation is responsible to deal with such degeneracies as will be discussed shortly.

Moving_LineString* and *Moving_Polygon

Moving_LineString is a moving type that is also constructed as a pointer to a nested table consisting of *Unit_Moving_LineString* objects. The difference between this moving type and the previously defined is that the *Unit_Moving_LineString* is also defined as a pointer to another nested table comprising of *Unit_Moving_Segment* objects. *Unit_Moving_Segment* in its turn is formed by three *Unit_Moving_Point* objects and a flag indicating the kind of interpolation between the starting and the 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 the *Unit_Moving_Segment* object. Figure 5 illustrates the structure of the *Moving_LineString* object.

The *Moving_Polygon* definition is very close to that of *Moving_LineString*. The main difference in the two definitions is on the construction of the corresponding unit moving type. More specifically, apart from a pointer to a collection of *Unit_Moving_ Segment* objects, the *Unit_Moving_Polygon* object has an additional attribute, a flag that indicates if this set of moving segments forms the exterior ring of a polygon or is an interior (hole) ring. In other words, this extra attribute adds the logic that disjoint moving holes may exist inside a moving polygon, with boundaries not crossing or touching the exterior boundary. Considering the rest aspects of the definition of *Unit_Moving_Polygon*, there is no difference between the two object types.

Moving_Collection* and *Moving_Object

Moving_Collection is the object type that models both homogeneous and heterogeneous collections of moving types. This is accomplished by defining it as a set of five pointers to each of the following types: *Multi_Moving_Point*, *Multi_Moving_LineString*, *Multi_Moving_Circle*, *Multi_Moving_Rectangle* and *Multi_Moving_Polygon*. Each of these moving types represents a homogeneous collection of moving points, linestrings, circles, rectangles and polygons,

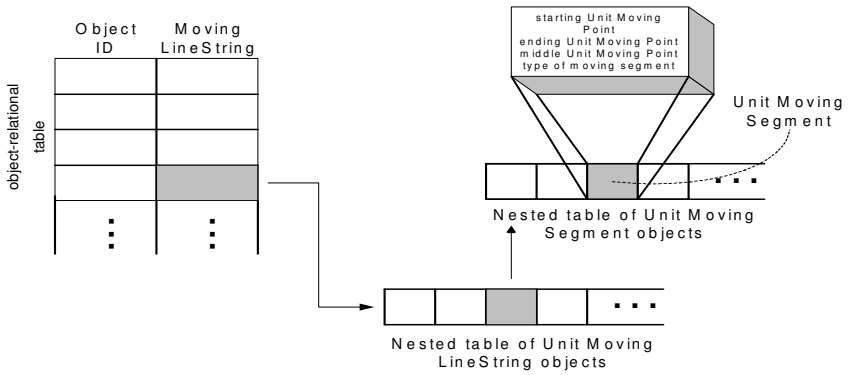

Figure 5 Structure of the *Moving_LineString* Object

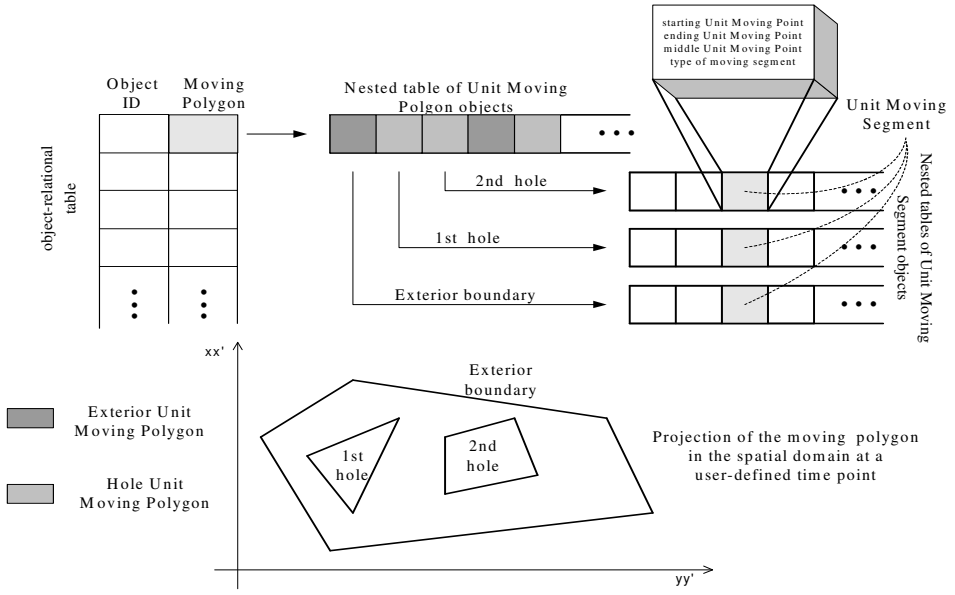

Figure 6 Structure of the *Moving_Polygon* Object

constructed as a pointer to a nested table of *Moving_Point*, *Moving_LineString*, *Moving_Circle*, *Moving_Rectangle* and *Moving_Polygon* object types, respectively. On the other hand, *Moving_Object* is the outcome of the conjunction of all the previous presented objects, and can be considered as the supertype of these types. Practically speaking, it is not intended to be directly used or constructed by the end-user. On the contrary, it is intended to be the result type of operations of the other moving types (i.e., system generated). As inferred from the structure of *Moving_Object* (cf. formula (23)), the pointers to the moving types presented in the preceding sections model the subtypes of the current type simulating inheritance.

OPERATIONS ON MOVING OBJECT DATA TYPES

Following, we classify the operations of the moving types introduced by HERMES into appropriate categories that enable us to describe and analyze the new query capabilities. The initial set of operations is the union of the methods supported by the simple moving types (namely, *Moving_Point*, *Moving_LineString*, *Moving_Circle*, *Moving_Rectangle*, *Moving_Polygon* and *Moving_Collection*). This set of operations is equivalent to the methods provided by the generic *Moving_Object* type as it models all the previous. The identifiable classes of operations that HERMES supports are:

i) *Consistency operations*: operations responsible for keeping the database in a consistent state (checking ordering and consecutiveness of periods of unit moving types, realizing degenerated cases, etc.).

ii) *Predicates*: operations that return boolean values concerning topological and other relationships between moving types (within distance, meet, overlap, etc.).

Table 1: Taxonomy of operations w.r.t. HERMES data types

Taxonomy of operations	Operations \ Data types	*Moving_object*	*Moving_Point*	*Moving_LineString*	*Moving_Circle*	*Moving_Rectangle*	*Moving_Polygon*	*Moving_Collection*
Database Consistency	*check_ordering*	√	√	√	√	√	√	√
	check_meet	√	√	√	√	√	√	√
	check_degeneracies	√	√	√	√	√	√	√
Topological Relationships	*f_within_distance*	√	√	√	√	√	√	√
	f_relate	√	√	√	√	√	√	√
Projection and Interaction to the Temporal and Spatial Domain	*unit_type*	√	√	√	√	√	√	√
	at_instant	√	√	√	√	√	√	√
	at_period	√	√	√	√	√	√	√
	f_buffer	√	√	√	√	√	√	√
	f_centroid	√			√	√	√	
	f_convexhull	√	√	√	√	√	√	√
	f_traversed	√	√	√	√	√	√	
	f_trajectory	√	√					
Set Relationships	*f_intersection*	√	√	√	√	√	√	
Numeric operations	*f_area*	√			√	√	√	
	f_length	√	√	√	√	√	√	
	f_speed	√	√					
	f_velocity	√	√					
	f_direction	√	√					

iii) *Projection operations*: operations that restrict and project moving types to temporal (e.g. at_instant, at_period) and spatial domain (e.g. trajectory, buffer).
iv) *Set operations*: basic set relationship operations (union, intersection, set difference).
v) *Numeric operations*: functions that compute a numeric value (e.g., the perimeter or the area of a moving polygon, the speed of a moving point).

The following sections describe the functionality of selected operations, representative of each class. The interested reader may find signatures and more algorithms in (Pelekis, & Theodoridis, 2010). A summary of the applicability of the proposed operators over the HERMES data types can be found in Table 1. Note that the last two classes of operators are defined only for *Moving_Point*, thus are omitted.

Maintaining Database Consistency

HERMES provides a set of object methods that enable the user to check the construction data of moving objects and maintain the database in a consistent state. These operations impose some integrity constraints that need to be followed for time-varying spatial data and, as such, protect the user from errors that have to do with the complex internal structure of the moving types. Below, three methods of this classs are discussed:

Op1. *boolean* ***check_ordering*** *():* there should be an ascending ordering of the periods between the unit moving types, each one represented by such a period. This mandatory constraint is required to model the evolution of the moving types in the timeline. The evolution of an object is represented by its consecutive unit moving types and the corresponding time periods should follow the same development.

Op2. *boolean* ***check_meet*** *():* checks if a period is consequent to the next period in the unit-type-order. The meaning of this operation is to assure that there is a smooth transformation of the time-changing geometries between sequential unit moving types and there are not *temporal gaps* between them.

Op3. *boolean* ***check_degeneracies*** *(Timepoint):* it is a method that checks if the geometry associated with a moving type at a specific time point is a non-degenerated geometry. Depending on the type, *check_degeneracies* imposes different restrictions on the development of these moving objects at user-defined time points. For *Moving_Point* there is not such an operation as there is no combination of mapped coordinates that could form an invalid geometry. For the rest of the *simple* moving types, the reader can find below some characteristic constraints enforced by HERMES:

Moving_LineString: (a) Checks if the *Unit_ Moving_Point* objects (two for line segments; three for

arc segments) that define the *Unit_Moving_Segment* objects become equal at a specific time point, thus degenerating a segment to a point; (b) Checks for overlapping between consequent *Unit_Moving _Segment* objects, meaning that the two time-varying coordinates of a *Unit_Moving_Point "fall"* upon the segment that is defined by the two previous *Unit_Moving_Point* objects; (c) Checks the coordinates of the starting *Unit_Moving_Point* of the first *Unit_Moving_ Segment not* to be equal at an instant, with the coordinates of the ending *Unit_Moving_Point* of the last *Unit_Moving_ Segment*. In such a situation, the potential *LineString* is degenerated to a *Polygon* geometry, regardless the fact that this polygon may have other anomalies (e.g. self-intersected segments that are acceptable in a *LineString* geometry); (d) in case of arc, it checks for co-linearity at a specific time point between the three *Unit_Moving_Point* objects that form the arc segment. In this case, the arc segment becomes a degenerated linear segment.

Moving_Circle: (a) Checks if the three *Unit_Moving_Point* objects that define a *Unit_Moving_Circle* object become equal at a specific time point, thus degenerating a circle to a point; (b) Assures that the three *Unit_Moving_Point* objects do not become co-linear.

Moving_Rectangle: (a) Checks if the lower left and upper right *Unit_Moving_Point* objects that define a *Unit_Moving_Rectangle* object become equal at a specific time point, thus degenerating a rectangle to a point; (b) Checks if x or y ordinates of the projected lower left and upper right *Unit_Moving_Point* objects become equal, meaning that the produced rectangle is collapsed to a linear segment parallel to *xx'* or *yy'* axis, respectively.

Moving_Polygon: (a) Checks for the same rules and constraints as in the case of *Moving_Linestring*, with the difference that, instead of inequality, it imposes equality between the starting and ending *Unit_Moving_Point*; (b) Checks if the *Unit_Moving_Polygon* objects that represent holes of a *Moving_Polygon* are always *"disjoint"* and *"inside"* the exterior boundary.

Figure 7 illustrates four degenerated cases for a *Moving_LineString* (that also stand for a *Moving_Polygon*, except case c), while those for a *Moving_Circle* and a *Moving_Rectangle* are illustrated in Figure 4. A complete description of all the degenerated cases as well as some interesting allowable cases is presented (Pelekis et al., 2010).

In the previous paragraphs, we described the operations concerning the constraints that should hold in a database of *simple* moving objects. The corresponding methods of a homogeneous or heterogeneous collection of such moving types, represented by the *Moving_Collection* object, follow a different strategy. In other words, these operations traverse one by one all the component objects of the *multi-* moving types that compose a *Moving_Collection* object, and apply the previous discussed operations to them. The first moving type that causes an error or is detected to be invalid or degenerated stops this process and informs the user with an appropriate message.

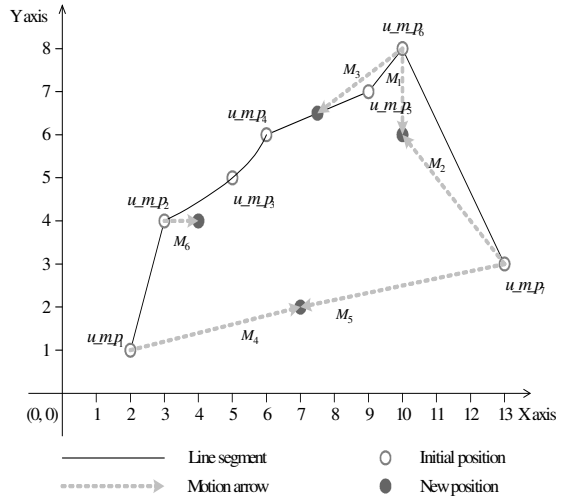

Figure 7 Four degenerated cases for a Moving_LineString: a) when $u\_m\_p_6$ and $u\_m\_p_7$ follow motions M_1 and M_2, respectively, b) when $u\_m\_p_6$ follows motion M_3, c) when $u\_m\_p_1$ and $u\_m\_p_7$ follow motions M_4 and M_5 respectively, d) when $u\_m\_p_2$ follows motion M_6.

Predicates Modeling Topological and Distance Relationships

HERMES provides object methods in the form of predicates to describe relationships between moving types. There are two sets of predicates supported by HERMES, namely *within_distance* and *relate*. Each set of predicates consists of eight operations, each of which models the relationship of the current moving type with a *Moving_Point*, *Moving_LineString*, *Moving_Circle*, *Moving_Rectangle*, *Moving_Polygon*, *Moving_Collection*, *Moving_Object* or *Geometry* object. Each operation comes 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. Also, many object methods in HERMES accept a *tolerance* parameter; if e.g. the distance between two points is less than or equal to the tolerance, HERMES 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. Below, we indicatively provide signatures of one

out of the eight operations, that for a *Moving_Polygon*. 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* argument from the parameter list. This is a common notation in the remainder of the paper.

Op4. *boolean {Moving_Object}* ***f_within_distance*** *(distance, Moving_Polygon, tolerance, Timepoint):* 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. Thus, the distance between two adjacent polygons is zero. On the other hand, the time-independent version 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 during which the two related objects are (or are not) within a specified Euclidean distance.

Op5. *Varchar {Moving_Object}* ***f_relate*** *(mask, Moving_Polygon, tolerance, Timepoint):* This generic predicate examines two moving objects and determines their topological relationship. As previously, it 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 (*"moving string"*), which describes the evolution in the topological relationship between the related objects. In particular, the *f_relate* operator implements the 9-intersection model for categorizing binary topological relations between moving geometries (Egenhofer, & Franzosa, 1991). Users can specify the kind of relationships that they require to check via the *mask* parameter.

Projection and Interaction to Temporal and Spatial Domain

HERMES provides object methods of special interest that have been proposed in the literature. Subsequently, we present the operations as these are defined for *Moving_Object* and the semantics behind these methods and we differentiate our presentation in case of change in the semantics of other moving types.

Op6. *Unit_Moving_Point* ***unit_type*** *(Timepoint):* This operation is the single method not defined for a *Moving_Object* type. Generally speaking, this operation is defined only for the *simple* moving objects that their construction is closely related with a collection of unit moving objects. The simple task that this function performs is that it finds (and returns) the unit-moving object whose attribute time period (*Period* object) *"contains"* the user-defined time point (*Timepoint* object). In other words, it returns that unit-moving type where the time instant represented by the argument *Timepoint* object is *"inside"* the time period that characterizes the unit-moving type. The *unit_type* method carries out all the necessary checks to maintain the database consistent and to ensure the validity of the moving object.

Op7. *Union_Output* ***at_instant*** *(Timepoint):* The *at_instant* operation is the most important method for the moving types introduced in HERMES, firstly because it is the operation that maps the unit function objects to spatial objects where moving objects reside at the given timepoint and, secondly, because it is the base of implementation for many other object methods. As already mentioned, the above signature concerns the *at_instant* operation for the *Moving_Object* type. 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 character string, respectively.

Op8. *Moving_Object* ***at_period*** *(Period):* The *at_period* object method is an operation that restricts the 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 all *Period* 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.

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

Op9. *Geometry {Moving_Object}* ***f_buffer*** *(distance, tolerance, Timepoint):* The *f_buffer* operation comes with two overloaded versions. The first generates a buffer polygon around a moving geometry object at a specific user-defined time point, while the second returns a *Moving_Object* modeling a time-

varying polygon, which describes a moving rounded buffer around a moving geometry. Obviously, this method is meaningless for a *Moving_Object* that represents a time-varying real number or string. The error handling mechanism of HERMES is responsible for realizing these situations and acting accordingly (e.g. by raising an appropriate error message).

The *f_buffer* operation for a homogeneous collection of moving geometries at a specific timepoint returns a multi-polygon where each polygon represents the buffer of its corresponding element in the collection. An interesting case is the buffer of a heterogeneous collection of moving objects, which is just one polygon that buffers all the different projected geometries together. The above-mentioned issues are visualized in Figure 8, where snapshots of different moving types and their corresponding buffer polygons are presented.

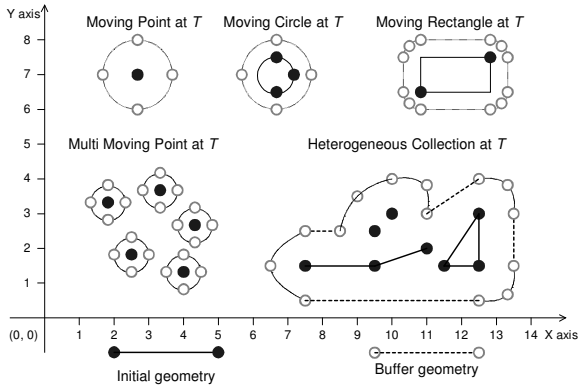

Figure 8 Demonstrating f_buffer operation

What is not illustrated in the description of the operation is the specific structure of these buffers for each corresponding moving type. Starting with the *Moving_Point*, someone would expect that the buffer of this type at a specific instant would be a circle geometry with radius the user-specified distance of the buffer. Surprisingly, the geometry returned by *f_buffer* operation is a polygon consisting of two arc segments that circle the point at the specified distance. The same happens in the case of the *Moving_Circle* where the buffer at a specific timepoint is defined as the buffer of its centre but the distance of the buffer is the initial user-specified distance plus the radius of the moving circle at that instant. The buffer of a *Moving_LineString*, a *Moving_Rectangle* and a *Moving_Polygon* at a specific timepoint is a compound polygon whose number of linear segments is equal to the number of linear segments that exist in the corresponding projected geometries and whose number of arc segments is equal to the number of vertices plus the number of arc segments.

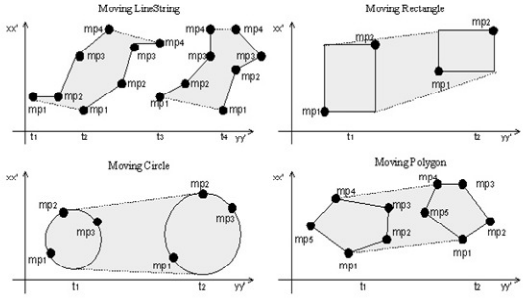

Figure 9 Areas Traversed by Moving Geometries

Op10. *Geometry {Moving_Object}* ***f_centroid*** *(tolerance, Timepoint):* The *f_centroid* operation returns the centre of a moving polygon object at user-defined time points. The centre is also known as the *"centre of gravity"*. The overloaded *f_centroid* function represents a moving point that at any time is the centre of gravity of the moving polygon object. The method is meaningful only for moving types that model single time-varying areas. In the rest cases (collections of moving geometries), an error message is raised by the error handling mechanism of HERMES.

Op11. *Geometry {Moving_Object}* ***f_convexhull*** *(tolerance, Timepoint):* The *f_convexhull* method returns a simple convex polygon that completely encloses the moving geometry object at a specific instant of time. The *Moving_Object* returned by the second version models a moving polygon that is the convex hull of a moving object at any time point (in other words, this is a convenient way to get an approximation of a complex geometry object).

Op12. *Geometry* ***f_traversed*** *():* The geometry returned by this function models all the places that a moving geometry *"traverses"* along its motion during the periods that characterize the unit moving objects. Such a geometry object is of polygon type. In the case of *Moving_Point* objects, the *f_traversed* method is transformed to a special operator (*f_trajectory*, to be discussed in the subsequent paragraph).

Figure 9 illustrates four examples of traversed areas, one for each of the simple moving types. In the case of the traversed *Moving_LineString*, we notice that the returned geometry is not a single polygon but a multi polygon due to the fact that the periods of the unit moving objects that compose the *Moving_LineString* do not *"meet"* each other or the variables that define the unit functions between subsequent unit moving objects present a substantial difference.

Op13. *Geometry* ***f_trajectory*** *():* This function is the *f_traversed* method for the case of a *Moving_Point* object. In other words, this operation simulates the trajectory traversed by a *Moving_Point*.

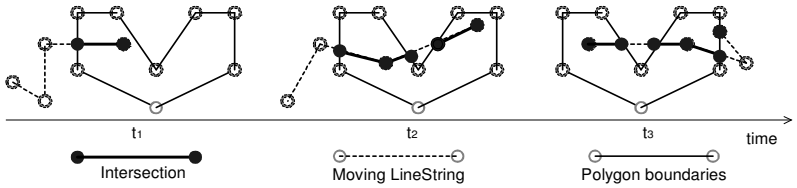

Figure 10 Demonstrating *f_intersection* Operation

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 beginning, ending and a random intermediate time instant of each one of the periods that identify the *Unit_Moving_Point* objects that compose the *Moving_Point*. Subsequently, the algorithm examines whether the intermediate projected coordinates "fall" upon the line formed by the other two pairs of coordinates. Depending on the result, a linear or arc segment connecting the beginning and ending projected coordinates is implied. A process of merging these segments follows, to form the returned *LineString* geometry.

Set Relationships

HERMES provides three object methods for describing set-relationships between moving types for intersection, union and set difference, respectively. Each comes with two overloaded versions, one for describing a geometry object as the result of applying the set-relationship at a user-defined time point and one for describing a moving geometry that is defined as the set-relationship at all the time periods that this relationship is meaningful. For example the intersection of a *Moving_Point* with a *Moving_Polygon* results in a *Moving_Object* that represents another moving point, which is the restriction of the initial *Moving_Point* inside or on the boundary of the *Moving_Polygon*.

Subsequently, we only present the intersection operation between any moving type and a *Moving_Polygon* object. Similar definitions exist for the rest two set relationships (union and set difference) and for all the other moving types, as well as for the respective operations describing set-relationships of a moving type with a pure spatial object.

Op14. *Geometry {Moving_Object}* ***f_intersection*** *(Moving_Polygon, tolerance, Timepoint):* The *f_intersection* object method returns either a geometry object that is the topological intersection (*AND* operation) of the two associated moving types projected at a user-defined time point or a *Moving_Object* whose mapping at each instant represents a geometry that is the outcome of this set operation.

Figure 10 depicts the instantiation of a *Moving_Object* modeling the intersection of a *Moving_LineString* with a polygon, at three different timepoints t_1, t_2, and t_3. At timepoint t_1 it is obvious that the result of the operation is a linestring geometry. At timepoint t_2 this intersection has as result a multi-linestring geometry due to the development of *Moving_LineString*, while at timepoint t_3 the resulted geometry is a heterogeneous collection of lines and points.

Numeric operations

HERMES supports a special class of object methods that either compute a numeric value of a moving object at a specific timepoint (e.g., the current perimeter of a moving polygon) or construct a *Moving_Object* representing the same time-varying numeric value. More specifically, we provide the subsequent numeric operations:

Op15. *number {Moving_Object}* ***f_area*** *(tolerance, Timepoint):* The *f_area* operation is defined for those moving types that their projection to the Cartesian plane depicts a closed region and computes the area for this region. The second (time-independent) version of the method returns a *Moving_Object* representing the time-varying area of a moving, extending and/or shrinking region. This function works with any moving polygon, including polygons with moving holes.

Op16. *number {Moving_Object}* ***f_length*** *(tolerance, Timepoint):* The *f_length* object method computes the length of a *Moving_LineString* object or the perimeter of a *Moving_Circle*, *Moving_Rectangle* or *Moving_Polygon* projected at the Cartesian plane at a user-defined time point. For a *Moving_Polygon* that contains one or more holes, this function calculates the perimeters of the exterior boundary and all holes at the given time point, and returns the sum of all the perimeters. The second version of the method returns a *Moving_Object* representing the time-varying length or perimeter of the moving type that invokes the operation.

Finally, as moving objects are time-dependent objects it would be useful to support operations that describe their rate of change. The only type that clearly qualifies the notion of derivation is the *Moving_Point* type. We define two operations called *speed* and *velocity*, respectively.

Op17. *number {Moving_Object}* ***f_speed*** *(Timepoint)*: The *speed* operation comes in two overloaded signatures. The time-dependent version returns a

number representing the *speed* of a moving point at a specific timepoint, while the time-independent version returns a *Moving_Object* modeling the time-varying *speed* at any time instant. The interested reader may find more numeric operations (such as *f_velocity*, *f_direction* etc.) in (Pelekis, & Theodoridis, 2010).

RELATED WORK

Several research efforts have tried to model spatio-temporal databases using the *moving object* concept. In (Erwig, Güting, Schneider, & Vazirgiannis, 1999) the authors propose a new line of research where moving points and moving regions are viewed as three-dimensional (2D + time) or higher dimensional entities whose structure and behavior is captured by modeling them as abstract data types. Such abstract data types for moving points and moving regions have been introduced in (Güting et al., 2000), together with a set of operations on such entities. The model presented in (Güting et al., 2000) was the first attempt to deal with continuous motion while in (Forlizzi et al., 2000) the definition of the discrete representation of the above-discussed abstract data types is presented. The interesting part of the discrete model is how *"moving"* types are represented. The authors describe the *sliced representation* behind which, the basic idea is to decompose the temporal development of a value into fragments called *"slices"* such that within the slice this development can be described by some kind of *"simple"* function. The next step in this development was the study of algorithms for the rather large set of operations defined in (Güting et al., 2000). Whereas (Forlizzi et al., 2000) just provides a brief look into this issue by presenting two example algorithms at the end, in (Lema et al., 2003) the authors present a comprehensive, systematic study of algorithms for a subset of the operations introduced in (Güting et al., 2000). Whereas some algorithms are relatively straightforward and simple, there are still a considerable number of quite involved ones. In all cases the authors analyze the complexity of the algorithms. In (Lema et al., 2003) the data structures from (Forlizzi et al., 2000) are also refined and extended by auxiliary fields to speed up computations. This paper also offers a blueprint for implementing such a *"moving objects"* extension package for suitable extensible database architectures. More specifically, the details and the current status of a prototypical implementation of the data structures and algorithms described are presented. The final outcome of this work has been recently demonstrated in (Almeida et al., 2006). The prototype is being developed as an algebra module for the experimental database system SECONDO (Dieker et al., 2000).

As an extension to the abstract model in (Güting et al., 2000), the concept of *spatio-temporal predicates* is introduced in (Erwig, Schneider, 2002). The goal is to investigate temporal changes of topological relationships induced by temporal changes of spatial objects. Further work on modeling includes (Su, Xu, & Ibarra, 2001) 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 (Becker, Blunck, Hinrichs, & Vahrenhold., 2004), a non-linear representation for moving objects is discussed in detail, while in (Vazirgiannis, & Wolfson, 2001) the authors consider movement in networks and some evaluation strategies.

Another model using moving objects is proposed in (Sistla, Wolfson, Chamberlain, & Dao, 1997; Wolfson, Sistla, Chamberlain, & Yesha, 1999; Wolfson, Xu, Chamberlain, & Jiang, 1998). 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. This model enables the DBMS to predict the future location of a moving object by providing a *motion vector*, which consists of its location, speed and direction for a recent period of time. In the model, the answer to a query depends not only on the database contents, but also on the time at which the query is entered. As long as the predicted position based on the motion vector does not deviate from the actual position more than some threshold, no update to the database is necessary. An important issue here is to balance the cost of updates against the cost of imprecise information. 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.

Related work in the field also includes our initial approach in designing HERMES. More specifically, in (Pelekis, Theodoridis, Vosinakis, & Panayiotopoulos, 2006) we briefly described the envisioned architecture of HERMES framework, in (Pelekis et al., 2006) we presented the primitives of the proposed datatype-oriented model and provides a preliminary insight on the supported functionality, while in (Pelekis, Frentzos, Giatrakos, & Theodoridis, 2008) we demonstrated the software developed theretofore, focusing in a specific (i.e. LBS) application domain. The current paper presents the complete system and describes all the necessary infrastructure for introducing our datatype system for moving objects. More specifically, we describe all the base, temporal and spatial types that compose the basic constructs for the definition of the moving objects datatypes, while we discuss in detail

the fundamentals for extending the previous with moving objects. In addition, all the datatypes, which are the core of the data type system of HERMES, are now formally defined and discussed in detail. The definition of the data type system is followed by a presentation of the design decisions and techniques for the physical representation of the proposed abstract data types. We further discuss the principles adhered by HERMES for designing moving objects operations and present in detail the full set of methods defined upon the proposed data types. Our design extends the data definition and manipulation language of OGC-compliant ORDBMS with spatio-temporal semantics and functionality. The proposed operations are accompanied with a discussion regarding their development and fruitful examples and illustrations for depicting the supported functionality. We also include a description of the implementation details of our system taking advantage of extensibility interfaces provided by state-of-the-art ORDBMS. Finally, we provide a qualitative comparison of our research effort with related work.

In (Güting, Behr, & Xu, 2010) the authors extended the SECONDO system with algorithms for efficient k-nearest neighbor search on moving object trajectories, while in (Güting, Behr, & Xu, 2010) they introduced a benchmark that defines datasets and queries for experimental evaluations. Another recent approach is TrajStore (Cudre-Mauroux, Wu, & Madden 2010), which focuses on supporting efficient spatio-temporal range queries in very large datasets.

In the following paragraphs we briefly present the differences of HERMES features proposed in this paper with the approach described in (Güting et al., 2000; Forlizzi et al., 2000) and (Lema et al., 2003), which is the most related to our work.

HERMES introduces time-varying geometries that change location or shape in discrete steps and/or continuously. Our approach for supporting both discretely and continuously changing spatio-temporal objects and which is based on the *Unit_Function* object is more generic and flexible than the tactic adopted in (Forlizzi et al., 2000) that asserts the same functionality. Apart from linear interpolations of spatial and spatio-temporal (moving) types utilized in (Forlizzi et al., 2000) and (Lema et al., 2003), HERMES also utilizes arc interpolations by proposing a categorization according to the quadrant the motion takes place and the motion heading. The user of HERMES is facilitated with a flexible and extensible interface for additional types of motion for moving types (e.g. splines, polynomials of degree higher than one etc.), which is provided via the *Unit_Function* object type.

In addition to *Moving_Point*, *Moving_LineString*, *Moving_Polygon*, proposed in (Forlizzi et al., 2000), the proposed *MOD Type System* also includes types like *Moving_Circle*, *Moving_Rectangle*, *Moving_Collection* and *Moving_Object*. A rich set of object methods is introduced that expresses all the interesting spatio-temporal phenomena and processes. This set of operations is a superset of the operations introduced in (Güting et al., 2000). The operation set commenced in (Güting et al., 2000) at an abstract level, is reduced in (Forlizzi et al., 2000) where specific finite representations and data structures are given for all the types of the abstract model, and is further reduced in (Lema et al., 2003) where a subset of the algorithms are selected to make the implementation manageable.

Of course, there are more differences between the two operations sets supplied by (Güting et al., 2000) and HERMES. For example, all topological operations introduced in (Güting et al., 2000) are combined in HERMES under a single operator, which distinguishes the different topological relationships via a *"mask"* parameter. Furthermore, HERMES introduces new operations describing the buffer, the convex hull, the centre of gravity and points on the surface of moving geometries. Additionally, particular attention has been paid to operations that facilitate the user to check the construction of moving objects and to keep such kind of spatio-temporal data in a consistent state. This leads to effective database maintenance and reliable error-handling mechanism.

The *Moving_Collection* object supports not only a homogeneous collection of moving types but also a heterogeneous collection of them. In (Güting et al., 2000), heterogeneous collections are not supported and a single moving type corresponds to a homogeneous *Moving_Collection* of the proposed *MOD Type System*. 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 (Güting et al., 2000) such degenerated moving types (moving reals, strings and booleans) are constructed as separate objects, which leads to a proliferation of object types that mainly are not spatio-temporal, which makes more difficult and unnatural the utilization of such data types by end users.

Generally speaking, the proposed *MOD Type System* is richer and more flexible than the one presented in (Güting et al., 2000). For example, it supports moving linestrings that intersect themselves during their development, while such a behavior is not allowed in (Güting et al., 2000) due to the fact that the spatial model does not accept self-intersecting linestrings. This is a very simple example of the importance that HERMES is OGC-compliant.

CONCLUSIONS AND OUTLOOK

In this paper, a formal framework and its implementation for managing and analyzing moving objects, called HERMES, was introduced. HERMES is a system extension that provides spatio-temporal functionality to OGC-compliant ORDBMS and supports modeling and querying of moving objects changing location either in discrete steps or continuously. A collection of data types and their corresponding operations are defined and implemented. Embedding the functionality offered by HERMES in ORDBMS data manipulation language provides a flexible, expressive and easy to use query language for moving object databases.

Another contribution of this work is that it prescribes straightforward future research directions. First of all, due to the fact that our study concerns only two-dimensional spatial objects as well as the change and motion of such geometries in the 2D plane, there is need to investigate the way we could model surfaces and three-dimensional spatial objects and the time-changing variants of them. Additionally, a future direction we are planning to follow is to utilize the optimization extensibility interface of existing ORDBMS in order to enhance the performance of HERMES. Finally, we will follow and extend the benchmark introduced in (Düntgen et al., 2009) for a more extensive comparison of HERMES with the approach of SECONDO.

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

Almeida, V.T., Güting, R.H., & Behr, T. (2006). Querying moving objects in secondo. In *Proceedings of the 7th International Conference on Mobile Data Management*.

Becker, L., Blunck, H., Hinrichs, K., & Vahrenhold., J. (2004). A framework for representing moving objects. In *Proceedings of DEXA*, (pp. 854-863).

Cattel, R.G.G., & Barry, D.K. (eds.). (1997, 05) *The object database Standard: ODMG 2.0.* Morgan Kaufmann Publishers.

Dieker S., & Güting, R.H. (2000). Plug and play with query algebras: Secondo. A generic dbms development environment. In *Proceedings of Int'l Symp. on Database Engineering and Applications* (IDEAS), (pp. 380-390).

Düntgen, C., Behr, T., & Güting, R.H. (2009). Berlinmod: a benchmark for moving object databases. *The VLDB Journal, 18*(6), 1335-1368.

Egenhofer, M., & Franzosa, R. (1991). Point-set topological spatial relations. *International Journal of Geographical Information Systems*, *5*(2), 161-174.

Erwig, M., Güting, R.H., Schneider, M., & Vazirgiannis, M. (1999). Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *GeoInformatica*, *3*(3), 265-291.

Erwig, M., & Schneider, M. (2002). Spatio-temporal predicates. *IEEE Transactions on Knowledge and Data Engineering*, *14*(4), 881-901.

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., Pelekis, N., Ntoutsi, I, & Theodoridis, Y. (2008). Trajectory database systems, In F. Giannotti and D. Pedreschi (eds), *Mobility, Data Mining and Privacy*. Springer.

Güting, R.H., Behr, T., & Xu, J. (2010). Efficient k-nearest neighbor search on moving object trajectories. *The VLDB Journal*.

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.

Güting, R.H. (1994). An introduction to spatial database systems. *The VLDB Journal*, *4,* 357-399.

Kakoudakis, I. (1996). The tau temporal object model. *MPhil Thesis, UMIST, Department of Computation.*

Koubarakis, M., & Sellis T. et al. (eds.). (2003). *Spatio-temporal databases: The Chorochronos Approach.* Springer.

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.

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., & Theodoridis, Y. (2006). Boosting location-based services with a moving object database engine. In *Proceedings of the 5th Int'l ACM Workshop on Data Engineering for Wireless and Mobile Access*.

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*.

Sistla, P., Wolfson, O., Chamberlain, S., & Dao., S. (1997) Modeling and querying moving objects. In *Proceedings of the 13th Int'l Conf. on Data Engineering*.

Su, J., Xu, H., & Ibarra, O. (2001). Moving objects: Logical relationships and queries. In *Proceedings of the 7th Int'l Symp. on Spatial and Temporal Databases*.

Tansel, A.U., Clifford, J., Gadia, S., Jajodia, S., Segev, A., & Snodgrass, R. (1993). *Temporal databases: Theory, design and implementation*. Benjamin/Cummings Publishing Company.

Theodoulidis I., & Loucopoulos, P. (1991). The time dimension in conceptual modeling. *Information Systems, 16*(3), 273-300.

Vazirgiannis M., & Wolfson, O. (2001). A spatiotemporal model and language for moving objects on road networks. In *Proceedings of the7th Int'l Symp. on Spatial and Temporal Databases*.

Wiederhold, G., Jajodia, S., & Litwin, W. (1991, 05). Dealing with granularity of time in temporal databases. In *Proceedings of the 3rd Nordic Conf. on Advanced Information Systems Engineering*.

Wolfson, O., Sistla, A. P., Chamberlain, S., & Yesha Y. (1999). Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, *7* (3), 257-387.

Wolfson, O., Xu, B., Chamberlain, S., & Jiang, L. (1998). Moving objects databases: Issues and solutions. In *Proceedings of the 10th Int'l Conf. on Scientific and Statistical Database Management*.