Abstract and Discrete Modeling of Spatio-Temporal Data Types*

Martin Erwig [1] Ralf Hartmut Güting [1] Markus Schneider [1] Michalis Vazirgiannis [2]

1) Praktische Informatik IV
Fernuniversität Hagen
D-58084 Hagen
GERMANY

2) Dept of Informatics
Athens Univ. of Economics & Business
Patision 76, 10434, Athens
GREECE (Hellas)

Abstract

Spatio-temporal databases deal with geometries changing over time. In general, geometries cannot only change in discrete steps, but continuously, and we are talking about moving objects. If only the position in space of an object is relevant, then *moving point* is a basic abstraction; if also the extent is of interest, then the *moving region* abstraction captures moving as well as growing or shrinking regions. We propose a new line of research where moving points and moving regions are viewed as three-dimensional (2D space + time) or higher-dimensional entities whose structure and behavior is captured by modeling them as abstract data types. Such types can be integrated as base (attribute) data types into relational, object-oriented, or other DBMS data models; they can be implemented as data blades, cartridges, etc. for extensible DBMSs. We expect these spatio-temporal data types to play a similarly fundamental role for spatio-temporal databases as spatial data types have played for spatial databases. In this paper we consider the need for modeling spatio-temporal data types on two different levels of abstraction.

1 Introduction

In the past, research in spatial and temporal data models and database systems has mostly been done independently. Spatial database research has focused on supporting modeling and querying of geometries associated with objects in a database [5]. 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*) [14]. Nevertheless, many people have felt that the two areas are closely related, since both deal with "dimensions" or "spaces" of some kind, and that an integration field of "spatio-temporal databases" should be studied and would have important applications. The question is, what the term *spatio-temporal database* really means.

Clearly, when we try an integration of space and time, we are dealing with *geometries changing over time*. In spatial databases, three fundamental abstractions of spatial objects have been identified: a *point* describes an object whose location, but not extent, is relevant, e.g. a city on a large scale map. A *line* (meaning a curve in space, usually represented as a polyline) describes facilities for moving through space or connections in space (roads, rivers, power lines, etc.). A *region* is the abstraction for an object whose extent is relevant (e.g. a forest or a lake). These terms refer to two-dimensional space, but the same abstractions are valid in three or higher-dimensional spaces.

Now, considering points, the usual word for positions or locations changing over time is *move*. Regions may change their location (i.e. *move*) as well as their shape (*grow* or *shrink*). Hence we conclude that spatio-temporal databases are essentially *databases about moving objects.*

Since lines (curves) are themselves abstractions or projections of movements, it appears that they are not the primary entities whose movements should be considered, and we should focus first on *moving points* and *moving regions*. In the approach described in this paper we will consider the fundamental properties of moving points and moving (and evolving) regions and try to support their treatment in data modeling and querying, rather than be driven by particular (existing) applications. On the other hand, if we succeed in providing such basic support, then we may be able to initiate applications that so far have never been thought of:

Migrations in the animal kingdom (e.g., whales, birds) can be appropriately modeled by moving points. Interesting queries ask for the trajectories of animal routes, the traversed distance, their speed, and the number and locations of their stops. Another example for moving points are transports (e.g., cars, planes, ships, buses, trains). Interesting queries inquire which taxi is closest to a passenger request position, which two planes are heading towards each other (going to crash), whether a plane crossed the air territory of some state X, or whether any ship is heading towards shallow areas.

Applications of moving regions are temporal evolutions of geographical entities (e.g., countries, lakes, forests, glaciers, continents). Here possible queries ask for the largest extent ever of the Roman empire, the occasions when two states merged, the extent and the rate of the Amazone rain forest

* This research was partially supported by the CHOROCHRONOS project, funded by the EU under the Training and Mobility of Researchers Programme, Contract No. ERB FMRX-CT96-0056.

shrinking, the location of glacier X at time Y, or the history of continental shift. Another example are climatic phenomena (e.g., storms, high/low pressure areas, temperature zones, cloud cover). Queries might ask for the direction of a tornado, its arrival at a specific location, and the development of air pressure areas.

Although we focus on the general case of geometries that may change in a continuous manner (i.e. move), one should note that there is a class of applications where geometries change only in discrete steps. Examples are boundaries of states, or cadastral applications, where e.g. changes of ownership of a piece of land can only happen through specific legal actions. Our proposed way of modeling is general and includes these cases, but for them also more traditional strategies could be used.

Also, if we consider transaction time (or bitemporal) databases, it is clear that changes to geometries happen only in discrete steps through updates to the database. Hence it is clear that the description of moving objects refers first of all to valid time. So we assume that complete descriptions of moving objects are put into the database by the applications, which means we are in the framework of historical databases reflecting the current knowledge about the past of the real world. Transaction time databases about moving objects may be feasible, but will not be considered initially.

There is also an interesting class of applications that can be characterized as artifacts involving space and time, such as interactive multimedia documents, virtual reality scenarios, animations, etc. The techniques developed here might be useful to keep such documents in databases and ask queries related to the space and time occurring in these documents.

The purpose of this paper is to describe and discuss an approach to modeling moving and evolving spatial objects based on the use of *abstract data types*. Essentially, we introduce data types for moving points and moving regions together with a set of operations on such entities. One needs also a number of related auxiliary data types, such as purely spatial or temporal types, time-dependent real numbers, and so forth. This collection of types and operations can then be integrated into any DBMS object algebra or query language to obtain a complete data model and query language.

The goal of this first paper on the approach is not yet to offer a specific design of such types and operations or a formal definition of their semantics. This needs to be done in further steps. Instead, here the goal is to give an outline of the work that should be done, in particular, we demonstrate the need for *two* related models on different abstraction levels. More design decisions are discussed in a longer version of this paper, which takes a broader point of view [4].

The paper is structured as follows: Section 2 explains the basic idea of spatio-temporal data types in a bit more detail. Section 3 and 4 contrast the continuous and discrete approaches to data modeling, a short concluding statement follows in Section 5. Section 6 discusses related work, and Section 7 offers general conclusions and future work.

2 The Basic Idea

Let us assume that a database consists of a set of *object classes* (of different *types* or *schemas*). Each object class has an associated set of *objects*; each object has a number of *attributes* with values drawn from certain *domains* or *atomic data types*. Of course, there may be additional features, such as object (or oid-) valued attributes, methods, object class hierarchies, etc. But the essential features are the ones mentioned above; these are common to all data models and already given in the relational model.

We now consider extensions to the basic model to capture time and space. As far as objects are concerned, an object may be created at some time and destroyed at some later time. So we can associate a validity interval with it. As a simplification, and to be able to work with standard data models, we can even omit this validity interval, and just rely on time-dependent attribute values described next.

2.1 Spatio-Temporal Types and Operations

Besides objects, attributes describing *geometries changing over time* are of particular interest. Hence we would like to define collections of *abstract data types*, or in fact *many-sorted algebras* containing several related types and their operations, for spatial values changing over time. Two basic types are *mpoint* and *mregion*. Let us assume that purely spatial data types called *point* and *region* are given that describe a point and a region in the 2D-plane[1] (a region may consist of several disjoint areas which may have holes) as well as a type *time* that describes the valid time dimension. Then we can view the types *mpoint* and *mregion* as mappings from time into space, that is

$$\underline{mpoint} = \underline{time} \rightarrow \underline{point}$$
$$\underline{mregion} = \underline{time} \rightarrow \underline{region}$$

More generally, we can introduce a type constructor τ which transforms any given atomic data type α into a type $\tau(\alpha)$ with semantics

$$\tau(\alpha) = \underline{time} \rightarrow \alpha$$

and we can denote the types *mpoint* and *mregion* also as $\tau(\underline{point})$ and $\tau(\underline{region})$, respectively.

A value of type *mpoint* describing a position as a function of time can be represented as a curve in the three-dimensional space (x, y, t) shown in Figure 1. We assume that space as well as time dimensions are continuous, i.e., isomorphic to the real numbers. (It should be possible to insert a point in time between any two given times and ask, e.g., for a position at that time.)

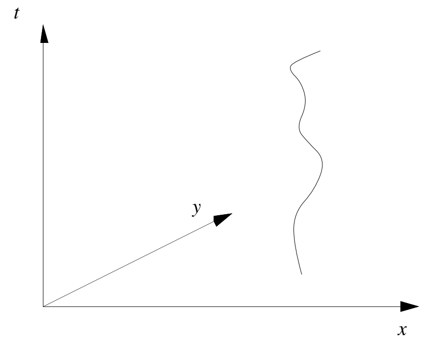

Figure 1: A moving point

A value of type *mregion* is a set of volumes in the 3D space (x, y, t). Any intersection of that set of volumes with a plane $t = t_0$ yields a *region* value, describing the moving region at time t_0. Of course, it is possible that this intersection is empty, and an empty region is also a proper region value.

[1] We restrict attention to movements in 2D space, but the approach can, of course, be used as well to describe time-dependent 3D space.

Generic operations for moving objects are, for example:

$\tau(\alpha) \times \underline{time}$	$\rightarrow \alpha$	**at**
$\tau(\alpha)$	$\rightarrow \alpha$	**minvalue**, **maxvalue**
$\tau(\alpha)$	$\rightarrow \underline{real}$	**duration**

At gives the value of a moving object at a particular point in time. *Minvalue* and *maxvalue* give the minimum and maximum values of a moving object. Both functions are only defined for types α on which a total order exists.

In particular, for moving spatial objects we may have an operation, such as

$\underline{mpoint} \times \underline{mpoint}$	$\rightarrow \underline{mreal}$	**mdistance**

Mdistance computes the distance between the two moving points at all times and hence returns a time changing real number, a type that we call <u>*mreal*</u> ("moving real"; <u>*mreal*</u> = $\tau(\underline{real})$).

Operations may also involve purely spatial or purely temporal types and other auxiliary types. For the following examples, let <u>*line*</u> be a data type describing a curve in 2D space which may consist of several disjoint pieces; it may also be self-intersecting. Let us also have operations

$\underline{mpoint}$	$\rightarrow \underline{line}$	**trajectory**
$\underline{line}$	$\rightarrow \underline{real}$	**length**

Here *trajectory* is the projection of a moving point onto the plane, and *length* returns the total length of a <u>*line*</u> value.

2.2 Some Example Queries

The presented data types can now be embedded into any DBMS data model as attribute data types, and the operations be used in queries. For example, we can integrate them into the relational model and have a relation

flights (id: <u>*string*</u>, from: <u>*string*</u>, to: <u>*string*</u>, route: <u>*mpoint*</u>)

We can then ask a query "Give me all flights from Düsseldorf that are longer than 5000 kms":

SELECT id
FROM flights
WHERE from = "DUS"
AND **length** (**trajectory** (flight)) > 5000

This query uses projection into space. Dually, we can also formulate queries projecting into time. For example, "Which destinations can be reached from San Francisco within 2 hours?":

SELECT to
FROM flights
WHERE from = "SFO" AND **duration** (flight) <= 2.0

Beyond projections into space and time, there are also genuine spatio-temporal questions that cannot be solved on projections. For example, "Find all pairs of planes that during their flight came closer to each other than 500 meters!":

SELECT A.id, B.id
FROM flights A, flights B
WHERE A.id <> B.id
AND **minvalue** (**mdistance** (A.route, B.route)) < 0.5

This is in fact an instance of a spatio-temporal join. Many more examples can be found in [4, 6].

3 Abstract Models Are Simple …

Abstract models allow us to make definitions in terms of infinite sets, without worrying whether finite representations of these sets exist. This allows us to view a moving point as a continuous curve in the 3D space, as an *arbitrary* mapping from an infinite time domain into an also infinite space domain. All the types that we get by applying the type constructor τ are functions over an infinite domain, hence each value is an infinite set.

This abstract view is the conceptual model that we are interested in. The curve described by a plane flying over space is continuous; for any point in time there exists a value, regardless of whether we are able to give a finite description for this mapping (or relation). In Section 2 we have in fact described the types mentioned under this view. In an abstract model, we have no problem in using types like "moving real", <u>*mreal*</u>, and operations like *mdistance*, since it is quite clear that at any time some distance between the moving points exists (when both are defined).

Defining formally an algebra for an abstract model looks as follows. We need to define carrier sets for the types (sorts) and functions for the operators. For a type t, we denote its carrier set as A_t and for an operator *op* the function giving its semantics as f_{op}. We consider the following example signature:

sorts <u>*point*</u>, <u>*time*</u>, <u>*mpoint*</u>, <u>*mreal*</u>
operators

$\underline{mpoint} \times \underline{time}$	$\rightarrow \underline{point}$	**at**
$\underline{mpoint} \times \underline{mpoint}$	$\rightarrow \underline{mreal}$	**mdistance**

We first define the carrier sets:

$$A_{\underline{point}} := \mathbb{R}^2 \cup \{\perp\} \qquad A_{\underline{time}} := A_{\underline{real}} := \mathbb{R} \cup \{\perp\}$$

So a point is an element of the plane over real numbers, or undefined.[2] For the "moving" types we can provide a single generic definition based on the type constructor τ:

$$A_{\tau(\alpha)} := \{f \mid f\colon A_{\underline{time}} \rightarrow A_\alpha \text{ is a function}\}$$

Functions are defined as follows. Let r, s be values of type <u>*mpoint*</u> and t a <u>*time*</u>. Furthermore, let $d(p, q)$ denote the Euclidean distance between two points in the plane.

$$f_{at}(r, t) := r(t)$$

$$f_{mdistance}(r, s) := g\colon A_{\underline{time}} \rightarrow A_{\underline{real}} \text{ such that}$$

$$g(t) = \begin{cases} d(r(t), s(t)) & \text{if } r(t) \neq \perp \wedge s(t) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

So abstract models are conceptually simple and their semantics can be defined relatively easily. Again, this simplicity is due to the fact that we admit definitions in terms of infinite sets and functions without worrying whether finite representations exist.

[2] We include the value $\perp$ (undefined) into all domains to make the functions associated with operators complete. This is more practical than have the system return an error when evaluating a partial function.

4 … But Only Discrete Models Can Be Implemented

The only trouble with abstract models is that we cannot store and manipulate them in computers. Only finite and in fact reasonably small sets can be stored; data structures and algorithms have to work with *discrete (finite) representations* of the infinite point sets. From this point of view, abstract models are entirely unrealistic; only discrete models are usable.

This means we somehow need discrete models for moving points and moving regions as well as for all other involved types (<u>*mreal*</u>, <u>*region*</u>, …). We can view discrete models as *approximations*, finite descriptions of the infinite shapes we are interested in. In spatial databases there is the same problem of giving discrete representations for in principle continuous shapes; there almost always *linear approximations* have been used. Hence, a region is described in terms of polygons and a curve in space (e.g. a river) by a polyline. Linear approximations are attractive because they are easy to handle mathematically; most algorithms in computational geometry work on linear shapes such as rectangles, polyhedra, etc. A linear approximation for a moving point is a polyline in 3D space; a linear approximation for a moving region is a set of polyhedra (see Figure 2). Note that a moving point can be a partial function, hence it may disappear at times, the same is true for the moving region.

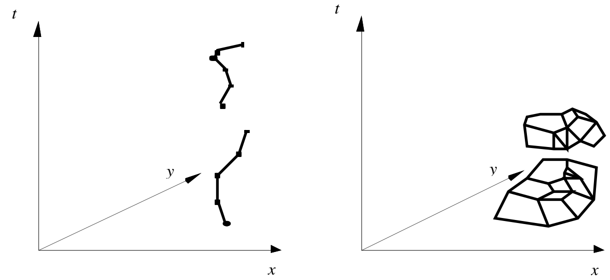

Figure 2: Discrete representations for moving points and moving regions

Defining formally an algebra for a discrete model means the same as for the continuous model: define carrier sets for the sorts and functions for the operators. We consider a part of the example above:

sorts <u>*point*</u>, <u>*time*</u>, <u>*mpoint*</u>
operators
$\underline{mpoint} \times \underline{time} \quad\quad \rightarrow \underline{point} \quad\quad$ **at**

Type <u>*mreal*</u> and operator *mdistance* have been omitted; for good reason, as we will see. Carrier sets can be defined as follows:

$A_{\underline{point}} := D_{\underline{point}} \cup \{\perp\}$ where $D_{\underline{point}} = \text{real} \times \text{real}$
$A_{\underline{time}} := D_{\underline{time}} \cup \{\perp\}$ where $D_{\underline{time}} = \text{real}$
$A_{\underline{real}} := \text{real} \cup \{\perp\}$

$A_{\underline{mpoint}} := \{<(p_1, t_1, b_1, c_1), \ldots, (p_m, t_m, b_m, c_m)> \mid m \geq 0,$
$(\forall\, i \in \{1, \ldots, m\}: p_i \in D_{\underline{point}}, t_i \in D_{\underline{time}}, b_i, c_i \in \text{bool}),$
$(\forall\, i, j \in \{1, \ldots, m\}: i < j \Rightarrow t_i < t_j)\ \}$

A few explanations are needed. Here by "real" and "bool" we mean data types offered by a programming language. We have introduced names for the defined part of a carrier set, e.g. $D_{\underline{point}}$. A moving point is represented by a sequence of quadruples. The sequence may be empty; this will mean that the position is undefined at all times. Each quadruple contains a position in space p_i and a time t_i. It also contains a flag b_i which tells whether the point is defined at times between t_i and t_{i+1} (b_i = true). This allows for the representation of partial functions (of the conceptual level). Finally, there is a flag c_i which states whether between t_i and t_{i+1} a stepwise constant interpretation is to be assumed, i.e., the point stayed in p_i, did not move (c_i = true), or linear interpolation, i.e., a straight line between p_i and p_{i+1}, is to be used (c_i = false). This representation has been chosen in order to support different classes of applications for moving point, e.g. unique events, stepwise constant locations, etc.

The intended meaning of the structure that we have just described needs of course to be formalized. This is exactly the semantics of the operator *at*:

> Let r be a value of type <u>*mpoint*</u> and t a <u>*time*</u> value. Let $r = <(p_1, t_1, b_1, c_1), \ldots, (p_m, t_m, b_m, c_m)>$ for some $m \geq 0$.
>
> $$f_{at}(r, t) := \begin{cases} \perp & \text{if } m = 0 \vee (m > 0 \wedge (t < t_1 \vee t > t_m)) \\ p_i & \text{if } m \geq 1 \wedge (\exists i \in \{1,\ldots,m\}:\ t_i = t) \\ lin(p_i,\ t_i,\ p_{i+1},\ t_{i+1},\ t) & \\ \quad \text{if } m \geq 2 \wedge (\exists i \in \{1,\ldots,m-1\} : (t_i < t < t_{i+1}) \wedge b_i \wedge \neg c_i) & \\ p_i & \text{if } m \geq 2 \wedge (\exists i \in \{1,\ldots,m-1\} : (t_i < t < t_{i+1}) \wedge b_i \wedge c_i \\ \perp & \text{if } m \geq 2 \wedge (\exists i \in \{1,\ldots,m-1\} : (t_i < t < t_{i+1}) \wedge \neg b_i \end{cases}$$
>
> where $lin(p_1, t_1, p_2, t_2, t)$ is a function that performs linear interpolation (puts a line through the two points (p_1, t_1) and (p_2, t_2) in 3D space and returns the point p on the line at time t).

One can observe that definitions for the discrete model are considerably more complex than those for the abstract model. On the other hand, they can be translated into data structures and algorithms which is not the case for the abstract model.

Apart from complexity, there are other difficulties with discrete modeling. Suppose we wish to define the type <u>*mreal*</u> and the operation *mdistance*. What is a discrete representation of the type <u>*mreal*</u> ? Since we like linear approximations for the reasons mentioned above, the obvious answer would be to use a sequence of pairs (*value*, *time*) and use linear interpolation between the given values, similarly as for the moving point.

If we now try to define the *mdistance* operator, we have to determine the time-dependent distance between two moving points represented as polylines. To see what that means, imagine that through each vertex of each of the two polylines we put a plane $t = t_i$ parallel to the xy-plane. Within each plane $t = t_i$ we can easily compute the distance; this will result in one of the vertices for the resulting <u>*mreal*</u> value. Between two adjacent planes we have to consider the distance between two line segments in 3D space. However, this is not a linear but a quadratic function (moving along the time axis, the distance may decrease and then increase again).

This is annoying, especially since the minimal distance between two moving points can be much smaller than the distance measured in any of the planes $t = t_i$. Hence using just these measurements as vertices for the moving real and then use linear interpolation would lead to quite wrong results. What can be done? One can either stick with linear interpolation and then add as vertices the focal points of the parabolas describing the time-dependent distance between two planes. In this way at least the minimal distance would not be missed. However, then the discrete model would already be inconsistent in itself, as the behavior of the distance between the two polylines is not correctly represented. An alternative would be to define the discrete model for the moving real in such a way, that it contains parameters for quadratic functions between two vertices. But this immediately raises other questions. Why just quadratic functions motivated by the *mdistance* operation, perhaps other operations need other functions? Should we allow parameters for polynomials? Up to what degree? Storing these parameters is expensive. And all kinds of operations that we need on moving reals must then be able to deal with these functions.

This example demonstrates what kind of nasty problems arise in discrete modeling that we simply do not see in abstract modeling.

5 Both Levels of Modeling Are Needed

We conclude that both levels of modeling are indispensable. For the discrete model this is clear anyway, as only discrete models can be implemented. However, if we restrict attention directly to discrete models, there is a danger that a conceptually simple, elegant design of query operations is missed. This is because the representational problems might lead us to prematurely discard some options for modeling.

For example, from the discussion above one might conclude that moving reals are a problem and no such type should be introduced. But then, instead of operations *minvalue, maxvalue*, etc. on moving reals one has to introduce corresponding operations for each time-dependent numeric property of a moving object. Suppose we are interested in *distance* between two moving points, *speed* of a moving point, and *size* and *perimeter* of a moving region. Then we need operators *mindistance*, *maxdistance*, *minspeed*, *maxspeed*, and so forth. Clearly, this leads to a proliferation of operators and to a bad design of a query language. So the better strategy is to start with a design at the abstract level, and then to aim for that target when designing discrete models.

6 Related Work

For several years researchers both in the spatial and in the temporal community have recognized the need of a simultaneous treatment and integration of data with spatial and temporal features in databases. A comprehensive bibliography on spatio-temporal databases until 1994 is given in [1]. Many of its articles document the interaction of space and time through application examples. But nevertheless, up to now research on models for spatio-temporal databases is still in its infancy.

Most of the research on this topic has focused on the extension of specialized spatial or temporal models to incorporate the other dimension. Most modeling approaches adopt the snapshot view, i.e., represent space-time data as a series of snapshots. Gadia et al. [7] propose time- and space-stamping of thematic attributes as a method to capture their time- and space-varying values. The time dimension describes when an attribute value is valid, and the spatial dimension expresses where it is valid. While each value has always a temporal evolution, it is doubtful whether it always has a spatial aspect. Worboys [15] defines spatio-temporal objects as so-called spatio-bitemporal complexes. Their spatial features are given by simplicial complexes; their temporal features are described by bitemporal elements attached to all components of simplicial complexes. In [3] and [12] event-based approaches for ST databases are proposed. Events indicate changes of the locations and shapes of spatial objects and trigger the creation of new versions in the database. All these approaches are only capable of modeling discrete or stepwise constant but not continuous temporal evolutions of spatial data.

Yeh and Cambray [16, 17] emphasize some aspects also mentioned in our paper. Since spatial data over time can be highly variable, they consider a continuous view of these data as indispensable and a snapshot view as inappropriate. So-called behavioral time sequences are introduced. Each element of such a sequence contains a geometric value, a date, and a behavioral function, the latter describing the evolution up to the next element of the sequence. Examples of such user or predefined functions are punctual functions, step functions, linear functions, and interpolation functions. A 2D object evolving in the course of time is described by a 3D object. While there are some similar ideas, they have no notion of abstract spatio-temporal data types with operations.

An interesting proposal that directly addresses moving objects is given in [13]. Here a moving object, e.g. a car or plane, is described by a so-called *dynamic attribute*. A dynamic attribute contains a *motion vector* and can describe the current status of a moving object (e.g. heading in a certain direction at a certain speed). An update to the database can change this motion vector (e.g. when a plane takes a turn). In this model a query will return different results when posed at different times; queries about the expected future are also possible. This model is geared towards vehicle tracking applications; in contrast to our proposal attributes do not contain the whole history of a moving object.

First attempts have been made to employ the constraint database approach [11] on spatio-temporal data. Work in constraint databases generally applies to spatio-temporal settings as arbitrary shapes in multidimensional spaces can be described. Two papers that explicitly deal with spatio-temporal examples and models are [9, 2].

7 Conclusions and Future Work

We have proposed a new approach to the modeling and implementation of spatio-temporal database systems based on spatio-temporal data types. This approach allows an entirely general treatment of time-changing geometries, whether they change in discrete steps, or continuously. Hence in contrast to most other work it also supports the modeling and querying of moving objects. Spatio-temporal data types can be used to extend any DBMS data model, and they offer a clear implementation strategy as extension packages to extensible DBMSs.

We feel that the paper opens up a new direction of research. As a first step, it is of crucial importance to clarify the underlying assumptions and to understand the available design options.

The next steps in this approach are the design of an abstract model, then a discrete model based on it, investigation of efficient data structures and algorithms for the discrete model, and implementation. We are currently completing the systematic design and formal definition of a system of data types and operations at the abstract level [6]. We plan to define a part of this design as a discrete model. Our own choice is to use linear descriptions for the *mpoint* and *mregion* types as well as for the spatial types (*line*, *region*) but to use (square roots of) quadratic functions for the representation of moving reals. In this way we can use the standard computational geometry algorithms for linear shapes, but have representations of time-dependent distances as well as perimeters and sizes of regions, that are consistent with the linear shapes on which they are based. As far as the design of data structures and algorithms and implementation are concerned, similar work has been done earlier for spatial databases in the ROSE algebra [10, 8].

Acknowledgments

We are grateful to our partners in the CHOROCHRONOS project for asking many of the questions addressed in this paper. Thanks for many clarifying discussions especially to Mike Böhlen, Christian Jensen, Nikos Lorentzos, and Timos Sellis.

References

[1] Al-Taha, K., R.T. Snodgrass, and M.D. Soo, Bibliography on Spatio-Temporal Databases. *ACM SIGMOD Record*, vol. 22, 59-67, 1994.

[2] Chomicki, J., and P.Z. Revesz, Constraint-Based Interoperability of Spatiotemporal Databases. *5th Int. Symp. on Large Spatial Databases*, 142-161, 1997.

[3] Claramunt, C., and M. Thériault, Managing Time in GIS: An Event-Oriented Approach. *Recent Advances in Temporal Databases*, Springer-Verlag, 23-42, 1995.

[4] Erwig, M., R.H. Güting, M. Schneider, and M. Vazirgiannis, Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. Technical Report 224, FernUniversität Hagen, Dec. 1997.

[5] Güting, R.H., An Introduction to Spatial Database Systems. *VLDB Journal 4*, 357-399, 1994.

[6] Güting, R.H., M.H. Böhlen, M. Erwig, C.S. Jensen, N. Lorentzos, M. Schneider, and M. Vazirgiannis, A Foundation for Representing and Querying Spatio-Temporal Data. Manuscript in preparation, 1998.

[7] Gadia, S.K., V. Chopra, and U.S. Tim, An SQL-Like Seamless Query of Spatio-Temporal Data. *Int. Workshop on an Infrastructure for Temporal Databases*, Q-1 - Q-20, 1993.

[8] Güting, R.H., Th. de Ridder, and M. Schneider, Implementation of the ROSE Algebra: Efficient Algorithms for Realm-Based Spatial Data Types. *4th Int. Symp. on Large Spatial Databases*, 216-239, 1995.

[9] Grumbach, S., P. Rigaux, and L. Segoufin, The DEDALE System for Complex Spatial Queries. *SIGMOD 98*, 1998, to appear.

[10] Güting, R.H., and M. Schneider, Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal 4*, 100-143, 1995.

[11] Kanellakis, P.C., G.M. Kuper, and P.Z. Revesz, Constraint Query Languages. *Journal of Computer and System Sciences 51*, 25-52, 1995.

[12] Peuquet, D.J., and N. Duan, An Event-Based Spatiotemporal Data Model (ESTDM) for Temporal Analysis of Geographical Data. *Int. Journal of Geographical Information Systems*, vol. 9, no. 1, 7-24, 1995.

[13] Sistla, A.P., O. Wolfson, S. Chamberlain, and S. Dao, Modeling and Querying Moving Objects. *IEEE Int. Conf. on Data Engineering*, 422-432, 1997.

[14] Tansel, A.U., J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass (eds.), Temporal Databases: Theory, Design, and Implementation. Benjamin/ Cummings Publishing Company, 1993.

[15] Worboys, M.F., A Unified Model for Spatial and Temporal Information. *The Computer Journal*, vol. 37, no. 1, 27-34, 1994.

[16] Yeh, T.S., and B. de Cambray, Time as a Geometric Dimension for Modeling the Evolution of Entities: A 3D Approach. *2nd Int. Conf. on Integrating GIS and Environmental Modeling*, 1993.

[17] Yeh, T.S., and B. de Cambray, Modeling Highly Variable Spatio-Temporal Data. *6th AustraliAsian Database Conf.*, 221-230, 1995.