STOQL: An ODMG-Based Spatio-Temporal Object Model and Query Language

Bo Huang[1] and Christophe Claramunt[2]

Department of Civil Engineering, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260, Tel: (65) 874-2158 Fax: (65)779-1635, cvehb@nus.edu.sg[1]
Naval Academy Research Institute, Lanveoc-Poulmic, BP 600, 29240 Brest Naval, France, Tel: 33 2 98 234206 Fax: 33 2 98 233857, claramunt@ecole-navale.fr[2]

Abstract

Spatio-temporal databases are a topic of growing research interest but there is still little work reported on spatio-temporal query languages in line with the Object Database Management Group (ODMG) standard. This paper presents the design of such a language, called STOQL, by adding spatial and temporal dimensions to OQL. This language is based upon a data model that extends the ODMG Object Model with basic spatial and temporal types as well as a parameterised type, **Temporal<T>**, which lifts spatial types into spatio-temporal types to support the representation of the history of an object. We show that STOQL can support the formulation of various spatio-temporal queries in relation to historical states of spatial objects as well as spatial changes including spatial type substitution.
Key words: spatio-temporal database, ODMG, object model, OQL, STOQL

1 Introduction

Spatial and temporal databases have long been studied on parallel, but relatively separate tracks. As natural and man-made objects are often associated with both space and time, there has been a growing interest in the integration of spatial and temporal components (Langran 1992, Worboys 1994, Claramunt and Thériault 1995, Peuquet and Duan 1995), leading to the emerging research field of spatio-temporal databases (Güting *et al.* 2000, Spery *et al.* 2001).

Compared with considerable research on temporal relational databases (e.g. Tansel *et al.* 1993, Snodgrass 1995a), a smaller amount of work has been reported in the context of temporal object databases (Snodgrass 1995b). This has similarly

occurred in spatio-temporal object databases. One of the important reasons has been the lack of an agreement for a common object model and query language (Huang *et al.* 2000). However, a consensus is now emerging, with the Object Data Management Group (ODMG) effort on standardising Object-Oriented Database (OODB) and a recent release of ODMG 3.0 (Cattell and Barry 2000). The proposed ODMG standard encompasses an object model (ODMG object model), an object definition language (ODL) and an object query language (OQL). This paper addresses the spatio-temporal data model and an associated query language. We propose the modelling of spatial and temporal data types using a novel parameterised temporal type **Temporal<T>** that lifts spatial types into spatio-temporal types to support the representation of the history of an object. The potential of this data type is illustrated by the application of spatio-temporal queries on a simplified case study.

Object-oriented databases (OODBs) provide powerful data abstractions and modelling facilities. The challenge is, however, to augment an object data model to capture the history of objects and to augment an object query language to permit queries over this history. The history could be represented at the object or attribute level (Fegaras and Elmasri 1998). As the attributes of an object may change asynchronously, attribute level time-stamping is preferred in this study. To represent the history of an attribute, we extend the ODMG Object Model with a parameterised type (a type generator just like **List<T>**), named **Temporal <T>**, which lifts a type **T** into a temporal type that contains the history of all states of **T**. The parameter type **T** can be any spatial type such as **Point**, **LineString**, **Polygon**, **Points**, **LineStrings** and **Polygons**, and so spatial types can be promoted to spatio-temporal types. Corresponding operations within the **Temporal** class are defined for traversing the history of objects. Thus, the implementation of temporal types is actually the implementation of the parameterised **Temporal** type, which, in turn, can meet the requirement of spatio-temporal data modelling and queries.

In this paper, we discuss the design of a spatio-temporal object query language, called STOQL, by extending OQL with some syntactic constructs to uniformly manipulate historical information of both spatial and non-spatial objects. The underlying model of STOQL establishes a parameterised type to support the definition of spatio-temporal types, facilitating the representation of the history of spatial objects and spatial changes including spatial type substitution. The reminder of this paper is organised as follows. Section 2 briefly summarises the principles of the underlying ODMG model and introduces the spatial, temporal and parameterised types. Section 3 describes how these data types can be manipulated at the query language level and illustrates the potential of our model using a simplified case study. Section 4 presents some related work. Finally Section 5 draws some conclusions.

2 Data Model

ODMG provides neither spatial types nor temporal types except Temporal Structured_literals (e.g., **Date**, **Time** and **Timestamp**). The objective of this paper is to extend the ODMG Object Model with spatial, temporal and parameterised types and to illustrate the interest of these extensions at the query language level. The ODMG standard provides an object definition language, ODL, for defining the object types that conform to the ODMG Object Model. STODL extends ODL with a number of class definitions and syntactical constructs that accommodate spatio-temporal information. STODL is hereafter used in class definitions.

2.1 Basic Spatial and Temporal Data Types

Spatial data types and operations have been defined in (OGC 1999). We primarily use the following geometric types: **Geometry**, **Point**, **LineString**, **Polygon**, **GeometryCollection**, **Points**, **LineStrings** and **Polygons**. **Geometry** is the root class of all these types, and is an abstract (non-instantiable) class. It includes operations that access properties of objects (e.g. **envelope**), determine topological relationship (e.g. **overlaps**, **meets**, and **disjoint**), and support spatial queries (e.g. **distance**, **buffer**, **intersection**, and **union**) (Figure 1). There are also other methods defined in the instantiable classes like **length** for the data type **LineString** and **area** for the data type **Polygon**; see (OGC 1999) for more details.

```
Class Geometry {
 //dimension 0-Point, Points
 //1-LineString, LineStrings
 //2-Polygon, Polygons
 attribute Unsigned Short dimension;

 //accessing spatial properties
 Struct MBR (Double minx; Double miny;
 Double maxx; Double maxy;);
 MBR envelope(in Geometry g);

 //test topological relationship
 Boolean disjoint (in Geometry g);
 Boolean intersects (in Geometry g);
 Boolean crosses (in Geometry g);
 Boolean contains (in Geometry g);
 Boolean overlaps (in Geometry g);
 …
 //create new spatial objects
 …
 Geometry intersection (in Geometry g);
 Geometry difference (in Geometry g);
}
```

```
Class TimeInterval {
  Attribute Timestamp start;
  Attribute Timestamp end;

  Unsigned Long duration();

  TimeInterval intersection (in
  TimeInterval tv);
  TimeInterval union (in TimeInterval tv);

  Boolean equals (in TimeInterval tv);
  Boolean contains (in TimeInterval tv);
  Boolean meets (in TimeInterval tv);
  Boolean overlaps (in TimeInterval tv);
  Boolean before (in TimeInterval tv);
  Boolean starts (in TimeInterval tv);
  Boolean finishes (in TimeInterval tv);
}
```

Fig. 1. The Geometry and the TimeInterval classes

In the context of temporal databases, two time dimensions, i.e., valid-time and transaction-time, are commonly represented (Jensen *et al.* 1992). Currently STODL only supports valid time although it can be easily extended to transaction time (valid-time is the time of a real-world event while a transaction-time gives the time of a database event). Based on the period definition in TSQL2 (Snodgrass 1995a) rather than the time interval definition in Allen (1983), the **TimeInterval** class is defined using STODL in Fig. 1.

The class **TimeInterval** has two attributes **start** and **end**, both of ODMG type Timestamp. Let **TI** be a variable of type **TimeInterval**. **TI** is constructed as **[TI.start, TI.end]**, which represents all the time points between and including **TI.start** and **TI.end**. If **TI.start** equals **TI.end**, **TI** contains just one time point, simply represented as **[TI.start]**. **[TI.start]** is treated as an equivalent of a time interval, i.e. **[TI.start, TI.start]**. This class defines a number of elementary temporal operations for comparing two time intervals such as **equals**, **contains** and **before**, as well as those for the intersection and union of two time intervals.

For a spatio-temporal data model, the critical types should include the integration of spatial and temporal types into spatio-temporal types. These are tackled by a parameterised type, or a type generator as described below.

2.2 The Temporal<T> Type

The ODMG Object Model includes type generators, collection types, and collection instances.

The collections supported by the ODMG Object Model include **Set<T>**, **Bag<T>**, **List<T>**, **Array<T>** and **Dictionary<T, V>**. Each of these is a type generator, parameterised by the type **T**. All the elements of a collection object are of the same type **T**. Note that collection type generators are represented as Template classes in the C^{++} binding of the ODMG standard.

In the class definition of these types (see Cattell and Barry, 2000 for more details), the ODL type **Object** has been chosen to represent type parameters. As a time-varying object can contain a list of historical states, and the value of this object can be of any type, a parameterised type (type generator), **Temporal<T>**, is needed to promote any type to a temporal type so that the history of this object can

```
Struct State {Object val; TimeInterval vt;};

interface TemporalFactory: ListFactory{
    Temporal new_of_size(in long size);
}
```

```
class Temporal: List {
    attribute temporal<t> value;

    //get entire history of the attribute
    List<State> getHistory();

    get state index number
    Unsigned Long getStateIndex(in Object val);
            raises (no_such_Tvalue)
}
```

Fig. 2. The Temporal<T> type

be represented and manipulated. The **Temporal** type, an extension of the **List** type, is specified and illustrated in Fig. 2.

In the ODMG Object Model, objects are created by invoking creation operations on **factory** interfaces provided on factory objects supplied to the programmer by the language binding implementation. The **new_of_size** operation in the **TemporalFactory** interface creates a collection with the given amount of initial storage allocated, where the given size is the number of elements for which storage is to be reserved.

A **Temporal** object is a temporally ordered collection of **val-vt** pairs:

$$\{(val_1, vt_1), (val_2, vt_2), \ldots, (val_n, vt_n)\}$$

where $val_1, \ldots, val_n$ are legal values of type **T**, $vt_1, \ldots vt_n$ are time intervals such that $vt_i \cap vt_j = \varnothing$, $i \neq j$ and $1 \leq i, j \leq n$, and each pair, i.e., (val_i, vt_i), is an instance of the **State** structure (Fig. 2). An attribute of the **Temporal <T>** type is called a *historical attribute*.

The parameter type **T** can be any ODMG type, and so a ODMG type is lifted into a temporal type. **T** can also be any spatial type, which lifts a spatial type (e.g., **Polygon**) into a spatio-temporal type (e.g., **Temporal <Polygon>**).

All the operations defined in the **List** are valid for **Temporal** types when a **State** is specified as the argument. In addition, the **Temporal** class defines two additional operations. The **getHistory** operation retrieves the entire history of a historical attribute, and the **getStateIndex** operation retrieves an index number associated with a state.

After accessing the entire history of a historical attribute, the value (**.val**), validtime (**.vt**) and index number of each state can be obtained. Operations on the value of a particular type, e.g., **Geometry**, can then be carried out according to application purposes.

2.3 Type Substitution

A spatial change implies a change of an object's spatial attribute value during a given time interval. Spatial objects may change their location, orientation, shape, and size over time, and so their topological relationships with other objects may also change.

Attributes, particularly spatial attributes, can change not only their values, but also their types within a supertype. For example, a fire projected on the 2D plane may be modelled as a point at time t_1, but as a polygon at time t_2, and finally as a polygons at time t_3. To tackle such a problem, an alternative is to use the supertype of relevant types, e.g. **Geometry** in this case. This benefits from the rich modelling capabilities of object-orientation. When instantiating the air pollution class, different geometric types such as **Point**, **Polygon** and **Polygons** are used such that the shape attribute of the air pollution is represented by

$$\{(\text{pointObj}, vt_1), (\text{polygonObj}, vt_2), \ldots, (\text{polygonsObj}, vt_n)\}$$

in which, pointObj, polygonObj, …, polygonsObj are instances of the **Geometry** type, and vt_1, vt_2, …, vt_n are time intervals of type **TimeInterval**.

In general, the key to handling type substitution lies in the definition of operations in a supertype that should be able to deal with different subtypes. In the above case, the operations defined in the **Geometry** type can be applied to these instances.

3 Query Language

Although there is no consensus for the taxonomy of temporal queries, some common requirements have been identified (Snodgrass 1995a) such as temporal selection, join and aggregation, as well as valid-time projection. Besides these, as spatio-temporal databases also target spatial changes over time, the spatio-temporal join (Güting *et al.* 2000) is incorporated in STOQL as well.

3.1 Syntactical Constructs in STOQL

In order to illustrate the expressiveness of the STOQL language, we introduce an example database that has been used in previous works to evaluate some geographical data models and query languages (Claramunt 1999, Spery *et al.* 2001). This example describes a land use database, relationships between some parcels and buildings, protected areas (whose areas have been extended over time) and a fire. The STODL specification of these classes is defined in Fig. 3.

In the following classes, the geometry of either a parcel, a building or a protected area may change over time. However, type substitution does not occur to these three classes**.** It might occur to the class **fire** under the **Geometry** type. Type substitution refers to the change of geometric type, in the context of our example a fire is first located by a point that approximates its location and then by a polygon that varies in time and eventually disappears.

class **parcel** (extent parcels key identifier)	class **building** (extent buildings key identifier)

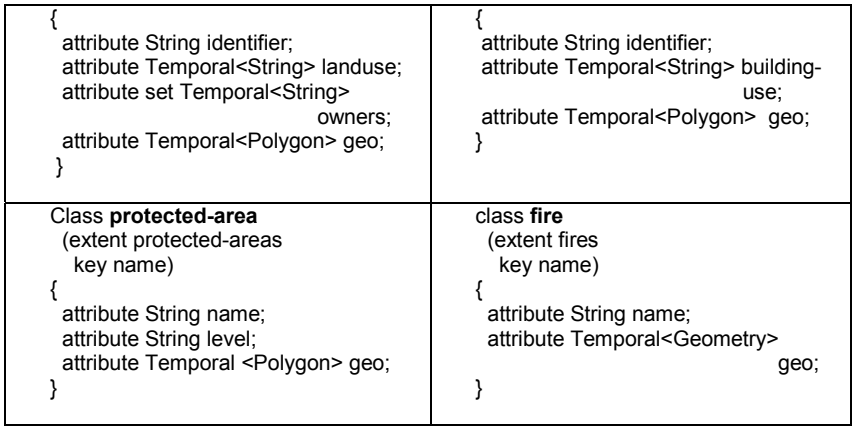

Fig. 3. Example database

STOQL extends OQL facilities to retrieve spatial temporal information. The states in histories are extracted through iteration in the OQL *from-clause*. Constraints in the *where-clause* can then be applied to the value, timestamp and index number of a state through corresponding operations that have been defined in the above model part. Finally the result is obtained through the projection operation in the *select-clause*. Clearly if the value is of a spatial type (i.e. **Point**, **LineString**, **Polygon**, **Points**, **LineStrings** and **Polygons**), then spatial operations will be employed.

Given the above principle, STOQL provides some syntactical extensions to OQL to traverse a history that is of type **Temporal<T>**. In Table 1, $\mathbf{time_1}$ and $\mathbf{time_2}$ are expressions of type **Timestamp**, **e** is an expression of type **Temporal<T>**, and **es** is an expression denoting a state within a history. These constructs facilitate the access to each state in a history.

Table 1. Syntactical constructs in STOQL

STOQL	OQL	Type
[$time_1$, $time_2$]	Struct(start: $time_1$, end: $time_2$)	TimeInterval
e!	e.getHistory()	List
es.val	es.val	T (any ODMG type and basic spatial types)
es.vt	es.vt	TimeInterval
es.index	e.getStateIndex(ev) (es in e)	Unsigned Long

Consider an example of class **parcel**. Let $\mathbf{parcel_1}$ be an object of the class **parcel**. Let **parcelgeo_3** be the fourth state in $\mathbf{parcel_1}$**.geo**'s history. $\mathbf{parcel_1}$**.geo!** refers to the entire history of $\mathbf{parcel_1}$**.geo**, which is, in fact, the whole list of $\mathbf{parcel_1}$**.geo**'s states. **parcelgeo_3.index** retrieves the index number of the state **parcelgeo_3**, i.e., 4 in this case. **parcelgeo_3.val** retrieves the geometric value of **parcelgeo_3**, and **parcelgeo_3.vt** the valid-time associated with **parcelgeo_3.val**.

The default value of **parcel$_1$.geo** is the value of **parcel$_1$.geo** at the current time (denoted as **[now]**), if no time value is given.

3.2 Query examples

The following examples illustrate different types of spatio-temporal queries such as temporal selection, index-based information retrieval, temporal and spatio-temporal joins, valid-time projection and type substitution. These queries are applied to self-explanatory classes. We assume that the granularity for all time-varying attributes in classes **parcel**, **building** and **protected-area** is the year, while the granularity for the geometrical attribute of the class **fire** is the day.

Query 1 (Temporal selection). Display graphically all the parcels of land use 'agricultural' that existed over the year 1980.

Select p-geo.val
From parcels As parcel, parcel.geo! As p-geo, parcel.landuse! As p-landuse
Where p-landuse.vt.contains([1980]) and
p-geo.vt.contains([1980]) and
p-landuse.val = 'agricultural'

In the above query, **parcel.geo!** returns the entire geometric history of **parcel**, and **parcel.landuse!** the entire landuse type history. Variables **p-geo** and **p-landuse** are states ranging over these two histories, respectively. The value of state **p-geo** can be viewed as a pair of a **Polygon** and **TimeInterval**. Similarly, the value of state **p-landuse** can be regarded as a pair of a **String** and **TimeInterval**. Time constraints on **p-geo** and **p-landuse** are respectively specified in the **Where** clause.

The next example illustrates how to retrieve the i^{th} state of an object, and how to synchronize events in STOQL (also called **temporal join**).

Query 2 (Index-based information retrieval and temporal join). Display graphically the first land use type of the parcel identified 'L1' from 1980 to 1990, and at that time where was the protected area of the river 'River1'.

Select p-landuse.val, p-areageo.val
From parcels As parcel, parcel.landuse! As p-landuse,
protected-areas As p-area, p-area.geo! As p-areageo
Where parcel.identifier = 'L1' and
p-landuse.index <=
All (**Select** ap-landuse.index
From parcel.landuse! ap-landuse
Where ap-landuse.vt.overlaps([1980, 1990])) and
p-areageo.vt.overlaps(p-landuse.vt) and
p-area.name = 'River1'

The first land use type of the parcel identified as 'L1' during a period may not simply be represented by '**p-landuse.index = 0**'. Instead it is the minimal index number during that period. Thus a subquery in the Where clause is employed. **p-**

areageo.vt.overlaps(p-landuse.vt) explicitly synchronizes events in relation to **p-landuse** and **p-areageo**. The valid-time of **p-areageo** is not necessarily [1980, 1990], but possibly a subset of this period, e.g. [1980, 1982].

As shown in the next example, spatial changes reflected in spatial properties and topological relationships of objects can be expressed by using two variables of the same class extent as well as different operations. This example also involves qualification over both space and time, and a **join** between different classes of objects.

Query 3 (Spatio-temporal join). Who were the owners of the parcels, which intersected the protected area of the river 'River1' over the year 1990, while they were away from that protected area over the year 1980.

```
Select   parcel.owners
From     parcels As parcel, parcel.geo! As parcelgeo1 parcelgeo2,
         protected-areas As p-area, p-area.geo! As p-areageo1 p-areageo2
Where p-area.name = 'River1' and
         p-areageo1.vt.contains([1980]) and parcelgeo1.vt.contains([1980]) and
         p-areageo1.val.disjoint(parcelgeo1.val)  and
         p-areageo2.vt.contains([1990]) and parcelgeo2.vt.contains([1990]) and
        p-areageo2.val.intersects(parcelgeo2.val)
```

As each state of an historical attribute can be accessed, spatial operations in the **Geometry** class such as **disjoint** and **intersects**, and temporal operations such as **contains** and **intersects** are employed naturally. In this sense, spatial and temporal operations are harmonised in the language.

Let us introduce a fourth query example that retrieves the time associated with a spatial change:

Query 4 (Validtime projection). When did the protected area of the river 'River1' first touched the parcel identified 'L1' ?

```
element
      (Select   (p-areageo.vt.intersection(p-geo.vt)).start
       From     protected-areas As protected-area,
                protected-area.geo! As p-areageo,
                parcels As parcel, parcel.geo! As p-geo
       Where    protected-area.name = 'River1' and parcel.name = 'L1' and
                p-areageo.val.touches(p-geo.val)
      ).requires[0]
```

As both the protected area and the parcel may spatially change, the **intersection** of the two relevant time intervals is used. The use of **element…requires[0]** obtains the first time that satisfies the conditions.

Temporal aggregation in STOQL is achieved using the standard OQL aggregators. The following query also uses the method **duration** to retrieve duration of a time interval before an aggregation:

Query 5 (Duration). For how long was the parcel identified 'L1' separated from the protected area of the river 'River1'?

```
Sum
(Select  p-geo.vt.duration()
 From    parcels As parcel, parcel.geo! As p-geo,
             protected-areas As p-area, p-area.geo! As p-areageo
 Where   parcel.identifier = 'L1' and p-area.name = 'River1' and
         p-geo.val.disjoint(p-areageo.val)
)
```

STOQL can also deal with queries involving spatial type substitution. A type casting might be used if necessary.

Query 6 (Type substitution). What is the total area of buildings that were affected by the area of the fire named 'Fire1' on May 25, 1992?

```
Sum
(Select  (Polygon)(f-geo.val).intersection(b-geo.val).area()
 From    buildings As building, building.geo! As b-geo,
         fires As fire, fire.geo! As f-geo
 Where   f-area.name = 'Fire1' and
         f-geo.vt.overlaps([25/5/1992]) and
         b-geo.vt.start = f-geo.vt.start.year() and
         b-geo.vt.end  = f-geo.vt.end.year() and
         b-geo.val.overlaps(f-geo.val)
)
```

This query returns the intersection area of the buildings with the fire identified 'Fire1' for a given time interval. The **b-geo.val.overlaps(f-geo.val)** expression in the **Where** clause evaluates if two objects overlap. If this relationship holds, the spatial **intersection** operation is performed, otherwise not. The **year()** operation assigns the valid-time of **f-geo** with the granularity of one year.

The use of **Polygon** casting before **f-geo.val** indicates that the compile-time type checker is told via the downcast that the intersection result must be of the **Polygon** type and the query is accepted as type correct. Note that at runtime, each occurrence of the intersection geometry in the select clause will be checked for its type.

4 Related Work

Previous work relevant to this paper is primarily associated with the **Temporal<T>** type and spatio-temporal object query languages. The notion of the **Temporal<T>** type was introduced in (Bertino *et al.* 1997), but there was no ODL-like definition for this type or a query language. Fegaras and Elmasri (1998) specified the definition of a similar type, which was, however, taken as a parametric class that is not supported by the current version of the ODMG standard (see also Alagic, 2001). Furthermore, since we apply this type in a

different query language, operations defined in this type are distinguished from theirs.

Güting *et al.* (2001) explored the representation and querying of moving objects by defining a set of Abstract Data Types (ADT), and ensuring consistency and closure of data types. They defined temporal types such as **mreal**, **mpoint**, **mpoints** and **mregion**. These types obtained through the moving type constructor are functions, or sets of pairs (instant, value). While these types are similar to type **Temporal <T>**, the former are instant based, and each temporal type like **mreal**, **mpoint**, **mpoints** and **mregion** requires a class definition, or code rewriting. In addition, this data model does not touch the representation and handling of spatial type substitution.

STOQL is motivated by the temporal object language proposed by Fegaras and Elmasri (1998), but differs greatly from the latter in the expression of time and temporal projections. In STOQL, such expressions are specified as time constraints in the **Where** clause, similar to the treatment of other constraints in OQL. STOQL also employs fewer syntactical constructs to make the language easier to understand especially for a novice user. More importantly, STOQL has attempted to express spatio-temporal queries and temporal queries in a uniform way. Spatio-temporal queries involve spatial properties, as well as spatial changes.

Cheng and Gadia (1993) have also proposed an object-oriented spatio-temporal structured query language (OOSTSQL), an extension of SQL, which supports both spatial and temporal data. OOSTSQL employs a proprietary clause, **Restricted to**, and temporal expressions to handle different constraints, and it focuses on temporal aspects rather than spatial aspects. Thus it does not address a rich set of spatial data types together with spatial operations.

Let us also mention the recent release of SQL-99 (formerly known as SQL 3). While OQL is an attempt to bring the best of SQL into the object-oriented world, SQL-99 can be characterised as bringing the best of object-orientation into the relational world (cf. Ullman and Widom 1997). Since we adopt an object database approach instead of object-relational, the ODMG standard is preferred in this study. There are also a number of standardisation efforts in the area of geographic and spatial information such as OGC (Open GIS Consortium), ISO TC211 and SQL/MM Spatial (Kottman 1998). Our extensions to the ODMG standard with respect to spatial and temporal operations have been addressed to comply with the core of these proposed standards.

5 Conclusion

This paper has presented an extension to the ODMG Object Model that incorporates spatial and temporal dimensions into database objects. The temporal extension is achieved by a parameterised type that is an orthogonal property of data types. On top of the model, we have also presented the design of a query language, STOQL, designed as a minimal extension to OQL. This supports an homogeneous manipulation of historical information for spatial and non-spatial

attributes. Benefited from the object-oriented paradigm, type substitution can also be represented and queried in STOQL.

In contrast to other related work, the distinct features of STOQL rely on that it builds upon existing standards (e.g. ODMG and OGC) and provides a flexible mechanism to deal with synchronous and asynchronous changes of spatial and aspatial properties. STOQL has been implemented on top of the commercial GIS package ArcView through an object-oriented scripting language. Further work concerns the support of transaction time and the integration of multiple time granularities.

Acknowledgements

Part of the work in this paper was done when the first author was working for the joint TRIPOD project between Keele University and Manchester University, UK. Discussions with Mike Worboys, Keith Mason, Chris Johnson, John Stell, Norman Paton, Alvaro Fernandes and Tony Griffiths were helpful in shaping the contribution of this paper. Part of the second author's work was carried out while he was working at the Nottingham Trent University.

References

Allen JF (1983) Maintaining knowledge about temporal intervals. Communications of the ACM 26: 832-843

Alagic S (1999) Type checking OQL queries in the ODMG type systems. ACM Transactions on Database Systems 24: 319-360

Bertino E, Ferrari E, Guerrini, G (1997) T_Chimera: a temporal object data model. Theory and Practice of Object Systems 3: 103-125

Cattell R, Barry D (eds) (2000) The Object Data Standard: ODMG 3.0. San Francisco, Morgan Kaufmann Publishers Inc.

Cheng TS, Gadia SK (1993) An object-oriented model for temporal databases. In: Snodgrass RT (ed) Proceedings of the International Workshop on an Infrastructure for Temporal Databases. Arlington, TX, June

Claramunt C, Thériault M (1995) Managing time in GIS: an event-oriented approach. In: Clifford J, Tuzhilin A (eds) Recent Advances in Temporal Databases. Springer-Verlag

Claramunt C, Parent C, Spaccapietra S, Thériault M (1999) Database modelling for environmental and land Use changes. In: Geertman S, Openshaw S, Stillwell J (eds) Geographical Information and Planning: European Perspectives. Springer-Verlag

Fegaras L, Elmasri R (1998) A temporal object query language. In: Fifth International Workshop on Temporal Representation and Reasoning (TIME-98), May 1998, Sanibel Island, Florida

Griffiths T, Fernandes A, Paton N, Mason K, Huang B, Worboys M, Johnson C and Stell J (2001) Tripod: a comprehensive system for the management of spatial and aspatial historical objects. ACM GIS 2001

Gueting RH, Böhlen MH, Erwig M, Jensen CS, Lorentzos NA, Schneider M, Vazirgiannis M (2000) A foundation for representing and querying moving objects. ACM Transactions on Database Systems 25: 1-42

Hornsby K, Egenhofer M (2000) Identity-based change: a foundation for spatio-temporal knowledge representation. International Journal of GIS 14: 207 – 224

Huang B, Worboys M, Johnson C, Stell J, Mason K (2000) A spatio-temporal object model and query language (extended abstract). In: GIScience 2000: First International Conference on Geographic Information Science. 28-31 Octover, Savannah, Georgia, USA

Jensen CS, Clifford J, Gadia SK, Segev A, Snodgrass RT (1992) A glossary of temporal database concepts. SIGMOD Record 21: 35-43

Kottman C (1998) Geoprocessing standards connections: OGC and TC/211, TC/204, SQL/MM, CORBAgis, W3C [online]. Availalbe from: http://www. opengis.org/info/newsletter/ 9810/16.htm.

Langran G (1992) Time in Geographic Information Systems. Taylor & Francis, London

Open GIS Consortium Inc. (OGC) (1999) OpenGIS Simple Features Specification for SQL, (Revision 1.1) [online]. Available from: http://www.opengis.org/techno /specs/99-049.pdf.

Peuquet D, Duan N (1995) An event-based spatio-temporal data model (ESTDM) for temporal analysis of geographical data. International Journal of GIS 9: 7-24

Snodgrass R (ed) (1995a) The TSQL2 Temporal Query Language. Kluwer Academic Publishers

Snodgrass R (1995b) Temporal object-oriented databases: a critical comparison. In Kim W (ed) Modern Database Systems: The Object Model, Interoperability, and Beyond. Addison-Wesley, pp 386-408

Spery L, Claramunt C, Libourel T (2001) A spatio-temporal model for lineage metadata. Geoinformatica 5:51-70

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

Ullman JD, Widom J (1997) A First Course in Database Systems. Prentice Hall Inc.

Worboys M (1994) A unified model for spatial and temporal information. The Computer Journal 37: 26-34