

Accepted for publication in *Transactions in GIS*, 2013

**An Algebra for Spatiotemporal Data: From Observations To
Events**

Karine Reis Ferreira¹, Gilberto Camara¹, Antônio Miguel Vieira Monteiro¹

¹DPI – Image Processing Division, INPE – National Institute for Space Research,
Av. dos Astronautas 1758, 12227-001 – São José dos Campos – SP – Brazil

(corresponding author: karine@dpi.inpe.br)

Abstract. Recent technological advances in geospatial data gathering have created massive data sets with better spatial and temporal resolution than ever. These large spatiotemporal data sets have motivated a challenge for Geoinformatics: how to model changes and design good quality software. Many existing spatiotemporal data models represent how *objects* and *fields* evolve over time. However, to properly capture changes, it is also necessary to describe *events*. As a contribution to this research, this paper presents an algebra for spatiotemporal data. Algebras give formal specifications at a high-level abstraction, independently of programming languages. This helps to develop reliable and expressive applications. Our algebra specifies three data types as generic abstractions built on real-world observations: *time series*, *trajectory* and *coverage*. Based on these abstractions, it defines *object* and *event* types. The proposed data types and functions can model and capture changes in a large range of applications, including location-based services, environmental monitoring, public health, and natural disasters.

Keywords: spatiotemporal data model, algebra, observations, fields, objects, events.

1 Introduction

The age of big geospatial data has come. Mobile phones, social networks and GPS devices create data useful for planning better cities, capturing human interactions and improving quality of life. Geosensors allow scientists to observe the world in novel ways. Space agencies worldwide plan to launch around 260 Earth observation satellites over the next 15 years. These massive data sets present a challenge for Geoinformatics. To use these large spatiotemporal data sets properly, we need innovative software designs. As a contribution to this design challenge, this paper presents an algebra for spatiotemporal data. The types and functions of the algebra can model data from many sources, including moving objects, remote sensing images, and geosensors.

Our model takes observations as a starting point, revisiting the classical work of Sinton (1978). This approach follows the ideas of Kuhn (2005): *“All information ultimately rests on observations, whose semantics is physically grounded in processes and mathematically well understood. Exploiting this foundation to understand the semantics of information derived from observations would produce more powerful semantic models”*.

The model is set forth as an algebraic specification, describing data types and operations in a language-independent and formal way. By separating specification from implementation, algebras help to develop reliable and expressive GIS applications (Frank et al. 1995; Frank 1999). Programmers can translate algebraic specifications into software using languages and environments of their choice. As an example, we have implemented the algebra using the open source TerraLib geospatial software library (Câmara et al. 2008).

2 Related work

To design spatiotemporal models, it is important to look at works that discuss change in objects (individual geographical units) and in fields (mappings from spatial locations to values). Relevant early results on object change include the bitemporal model of Worboys (1994) and the three-domain model of Yuan (1999). These models track

changes on the boundaries and attributes of an object, keeping its identity. These models have been extended by works such as Hornsby et al. (2000), who present a change description language with operations like ‘create’, ‘destroy’ and ‘continue existence’. Recent growth of mobile computing inspired much work on moving objects, notably the foundational algebra of Güting et al. (2000). Interest on location-based applications led to an ISO (2008) standard that defines a *moving feature* as an object whose geometry moves as a rigid body.

As to change in fields, Peuquet et al. (1995) propose a model that groups changes in raster cells by time of occurrence. Liu et al. (2008) introduce the idea of a *general field* with three spatial plus one temporal dimension to generalise previous definitions of fields. Mennis (2010) extends the conventional map algebra to include three-dimensional space and time. Efforts on standardisation led to the OGC *coverage* definition (OGC 2006). A *coverage* associates positions in a spatial, temporal or spatiotemporal domain to attribute values.

A further line of research is that of geospatial ontologies, that group real world phenomena in *continuants* and *occurrents* (Galton 2008). Continuants are entities whose identities remain constant as they undergo change, such as an aircraft and a volcano. Occurrents are entities that happen or occur, like a flight and an eruption. On the geospatial domain, ‘objects’ and ‘fields’ are taken as continuants and ‘events’ as occurrents (Galton et al. 2009). In this view, modelling only objects and fields misses part of the semantics of change. One also needs to consider events and the relations between events and objects (Worboys 2005). Following these ideas, Worboys et al. (2004) propose a model combining objects and events, defining event-event and event-object relations. Galton et al. (2005) refine these relations for events, states, and processes in dynamic networks. Hornsby et al. (2007) model events associated to moving objects and propose an approach to extract patterns of movements from them.

In this paper, we put together ideas from these three areas, proposing an algebra that represents objects, fields and events. We argue there are three key data types for spatiotemporal data: *time series*, *trajectory*, and *coverage*, from which we can derive the

object and *event* types. Using this step-by-step approach, the resulting algebra is useful for building many different applications.

3 From observations to events

We start with observations, our means to assess spatiotemporal phenomena in the real world (Kuhn 2009). According to Sinton (1978), there is an inherent structure to geographical information. For him, an observation should have three attributes: space, time and theme (the term “theme” refers to the real-world phenomenon or object being observed). He argues that we can create generalizations of geographical information based on *how* these attributes (space, time and theme) are assessed. In a general way, we observe the world by *fixing* one attribute, *controlling* another and *measuring* the other. Our observations are obtained by: (1) keeping one attribute constant; (2) varying the second attribute in a controlled way; and (3) measuring the third attribute, given the constraints of the second attribute. This produces six possible combinations. We consider that three of those are necessary and sufficient to model spatiotemporal data:

- 1) Fixing space, controlling time, and measuring theme results in a *time series*.
- 2) Fixing theme, controlling time, and measuring space results in a *trajectory*.
- 3) Fixing time, controlling space, and measuring theme results in a *coverage*.

The other three possible combinations are:

- 4) Fixing time, controlling theme, and measuring space.
- 5) Fixing space, controlling theme, and measuring time.
- 6) Fixing theme, controlling space, and measuring time.

As an example of combination (4), Sinton proposes a “vegetation map” created by finding out all locations of a given land cover type. However, these maps are more likely produced by a systematic data collection over a given area, resulting in *coverages*. Combination (5) occurs in cases like “measuring arrival times by runners in a marathon”. Yet it is also possible to get this type of data by analysing *trajectories* of runners. Sinton suggests “tide tables” as an example of combination (6). Since such tables can be obtained from *time series* that maps times to tide heights at a specific

location, there is no need for an additional type. Thus, using Occam’s razor, only three data types (*time series*, *coverage*, and *trajectory*) are needed to model all combinations of theme, time and space obtained by *fixing* one attribute, *controlling* another and *measuring* the third.

3.1 Data Abstractions

Using the *time series*, *trajectory*, and *coverage* types, we can define different views on the same observation set, meeting application needs. Take Figure 1 that shows the tracks of three cars equipped with GPS and air pollution sensors in a city. These cars produce a set of observations, each one containing a car identity, a time instant, a location and an air pollution value. Suppose the observations are collected hourly during one day. From this data it is possible to extract three different representations. Taking one of the cars as a sensor and the city as a spatial reference, we can build *time series* that shows the hourly air pollution in the city. Considering each car an individual object, we can get a set of *trajectories*. Fixing the whole day as a time reference and taking all air pollution data, we can create a *coverage* that conveys how pollution varied within the city during that day.

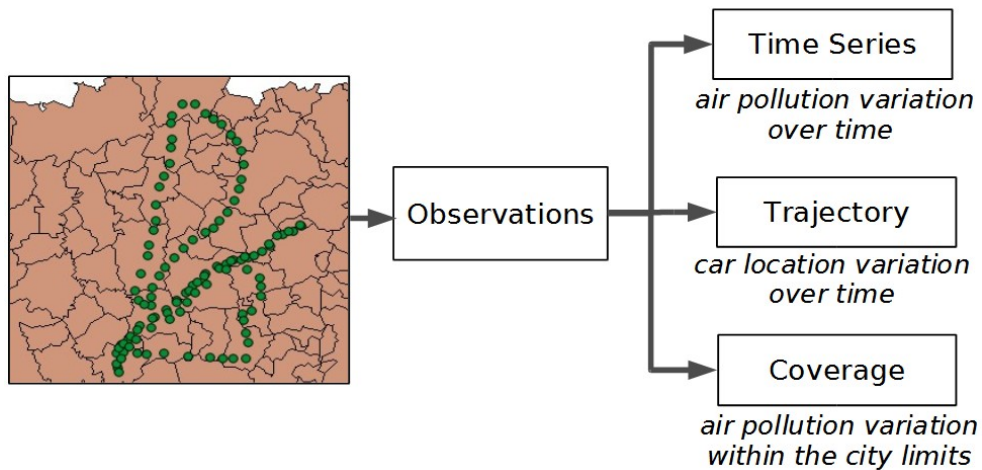


Figure 1. Different views on observations produced by moving cars.

A *time series* represents the variation of a property over time in a fixed location. Figures 2(a) and 2(b) show time series used in disease surveillance of dengue in the city

of Recife in Brazil (Regis et al. 2009). Dengue is a viral disease transmitted by mosquitoes. These mosquitoes lay their eggs in standing water; the eggs hatch in hot weather. To assess dengue risk, health services use buckets of water as egg traps. Figure 2(a) shows five meteorological stations and one of the associated temperature time series. The second set of time series shows the number of mosquito eggs gathered weekly from the egg traps. Figure 2(b) presents egg traps (red points) in a district of Recife and a time series produced by one of them.

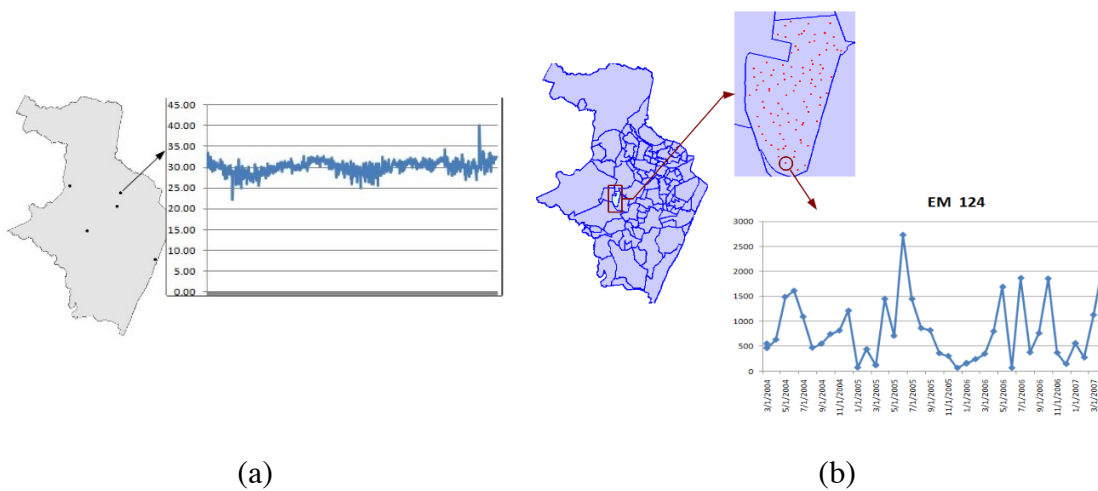


Figure 2: Examples of time series: (a) temperature collected by meteorological stations and (b) number of mosquito eggs gathered from one egg trap in a district of Recife, Brazil.

A *trajectory* represents how locations or boundaries of an object evolve over time. Figures 3(a) and 3(b) show trajectories. Figure 3(a) presents routes of sea elephants in Antarctica. Figure 3(b) shows the evolution of three city limits in the Brazilian state of Rondonia from 2001 to 2005.

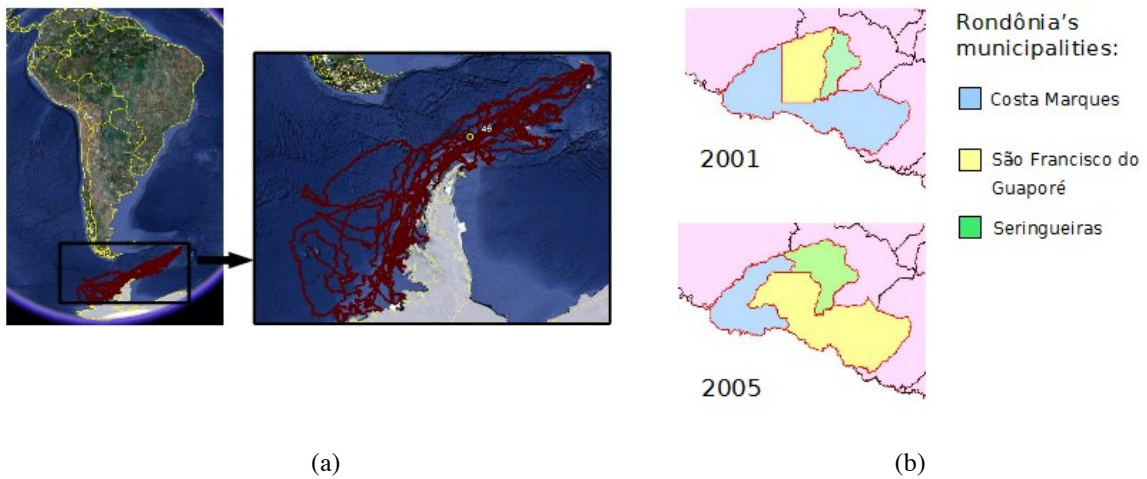


Figure 3: Examples of trajectories: (a) tracking of sea elephants in Antarctica and (b) evolution of three Rondônia's municipality limits during 2001 and 2005.

A *coverage* represents the variation of a property within a spatial extent at a time. Putting together the air pollution observations obtained by all cars of Figure 1 produces a coverage that shows how pollution varies in the city during one day. Other examples of coverages appear in Figure 4, which shows grids with the rain variation in the state of Rio de Janeiro during the natural disaster of 11 January 2011. The examples in the paper assume we have grids in 15-minute intervals. Figure 4 also shows the cities of the state of Rio de Janeiro, which will be used in the examples of events.

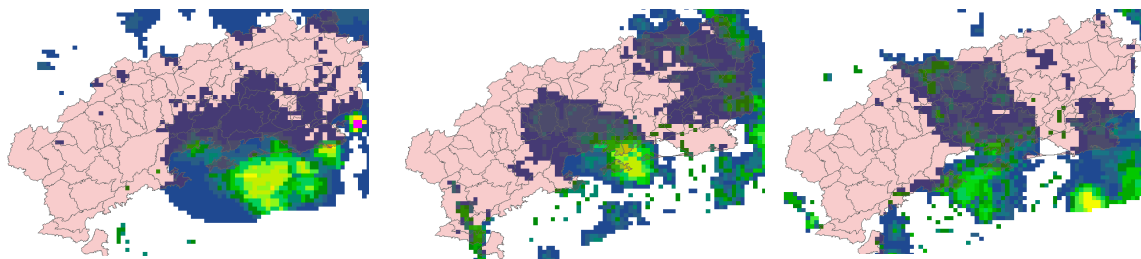


Figure 4: Example of coverages: rain in the state of Rio de Janeiro, Brazil, in 11 January 2011.

Since observations are discrete, they need to be combined with interpolation functions to approximate continuous change. Interpolators estimate values at locations in space and moments in time for which there is no data. Consider two observations of a moving car (Figure 1), one at instant 4 and the other at 8, shown in Figure 5(a). There are different methods to estimate car location at the non-observed time 6. Choices include a linear interpolator (Figure 5(b)) or a method that uses a street map as a spatial constraint, as in Figure 5(c). The proposed algebra allows choosing the most suitable interpolation function for each case.

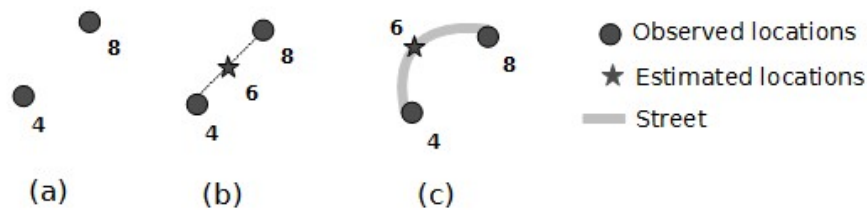


Figure 5. Observations of a moving car and different kinds of interpolation functions.

3.2 Objects and Events

Our model defines objects as *continuants* and events as *occurrents*. An object is an identifiable entity whose spatial and non-spatial properties can change over time. It is present as a whole at each moment of its existence (Galton et al. 2009). Examples of objects are cars (Figure 1), egg traps (Figure 2), sea elephants and municipalities (Figure 3) and cities of the state of Rio de Janeiro (Figure 4). An event is an individual episode with a definite beginning and end. It only exists as a whole across the interval over which it occurs. An event does not change over time. It can involve one or more objects, and an object can be involved in any number of events (Galton et al. 2009). In our model, we can derive events from specific conditions of spatial and non-spatial properties of objects. If we know what conditions lead to an event, we can express events using operations over the proposed types.

Consider the following objects: the cities of Rio and Recife and a group of sea elephants. A ‘flood’ event occurs in Rio if “rain is more than 10 mm/hour for more than 5 hours”. A ‘dengue epidemic’ event happens in Recife when “the average temperature is above 30^o C for more than a week and more than 50 eggs on average were found in the egg traps in the same week”. A ‘meeting of two animals’ occurs when “the minimal distance between two sea elephants is less than 2 meters”. These constraints are expressed through operations on *time series*, *trajectories* and *coverages*, which in turn are built from observations (Figure 6).

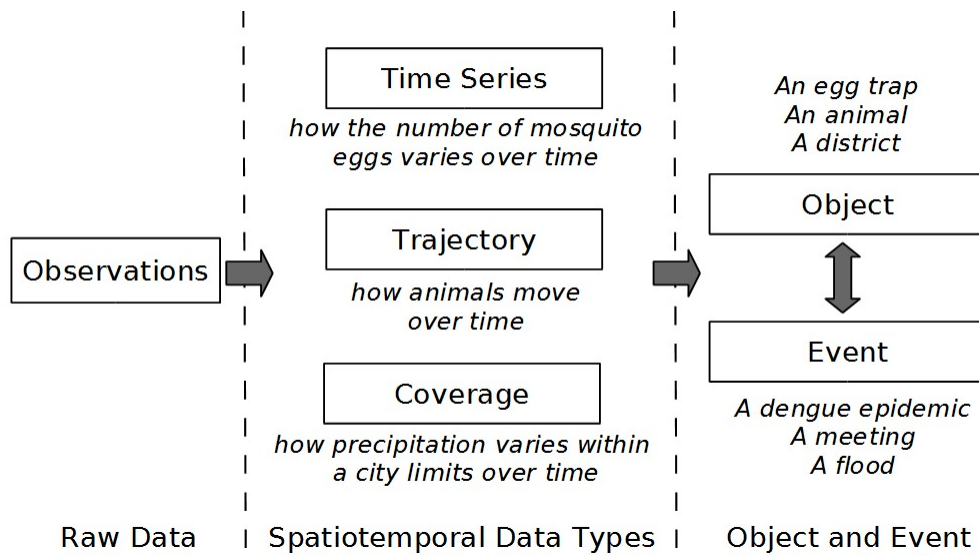


Figure 6. The proposed model.

4 An Algebra for Spatiotemporal Data

We use data types to express our abstractions. A *data type* is a set of values and a collection of operations on those values that defines their behaviour. An algebraic specification of a data type T consists in: (1) a *syntactic* description which defines the names, domains, and ranges of the operations of T ; and (2) a *semantic* specification which contains a set of axioms in the form of equations which relate operations of T to each other (Guttig et al. 1978). In what follows, functions and type signatures use monospaced font. Type names are given in TitleCase and function names in lowercase. Sets are enclosed by curly braces and square brackets denote parameterized types.

4.1 Primitive data types

There are three primitive types: Value, Time and Geometry. Value is a generic type to express attribute values that can be Integer, Float, String or Boolean. Typical operations on Value include `less_than`, `greater_than`, `equal_to`, `max`, and `min`. The meaning of such operations is evident when applied to numerical types. When applied to textual and boolean types, we consider the alphabetical order.

Time is a generic type that can be an Instant or a Period. The types Time, Instant and Period match the types *TM_GeometricPrimitive*, *TM_Instant* and *TM_Period* defined by the ISO temporal model (ISO 2002). Operations on Time include `equals`, `before`, `after`, `begins`, `ends`, `during`, `contains`, `overlaps`, `meets`, `overlappedBy`, `metBy`, `begunBy` and `endedBy`. They compare two time instances based on the temporal relationships of Allen (1983). Their behaviour when applied to instants and periods is described in the ISO standard (ISO 2002). Chronon is a generic type to represent temporal resolutions.

Geometry is a generic type compliant with the *Geometry* type defined in the OGC Geometry Model (OGC 2006). It can be a Point, Line, Polygon, MultiPoint, MultiLineString, or MultiPolygon type. Operations on Geometry include `equals`, `touches`, `disjoint`, `crosses`, `within`, `overlaps`, `contains` and `intersects`, as defined by OGC (2006). The types are:

Number:	Integer, Float
Value:	Number, String and Boolean
Time:	Instant, Period
Chronon:	Year, Month, Week, Day, Minute, Second.
Geometry:	Point, Line, Polygon, MultiPoint, MultiLineString, MultiPolygon.

We also define a null type, `Null`, to represent invalid values. In what follows, we omit the null type in the function signatures for clarity. Functions can return `Null` types in some cases, as described in the axioms. This behaviour should be considered when implementing the algebra.

4.2 Observations

type **Observations** [F:Type, C:Type, M:Type]

operations:

new: $\{(F,C,M)_1, (F,C,M)_2, \dots, (F,C,M)_n\} \rightarrow \text{Observations} \mid n > 0$

reference: $\text{Observations} \rightarrow F$

positions: $\text{Observations} \rightarrow \{C_1, \dots, C_n\}$

measure: $\text{Observations} \times C \rightarrow M$

An observation is a tuple of three attributes: time (Time), location (Geometry) and value (Value). The **Observations** type has three type parameters. Following Sinton (1978), the first type is the fixed reference (F), the second is the controlled attribute (C) and the other is the measured attribute (M). The constructor **new** builds an observation set from a set of instances of types F, C and M. **Reference** returns the value of the fixed attribute. The **positions** function reports the variation of the controlled attribute and **measure** returns the observed value associated to a position.

4.3 Interpolator

type **Interpolator** [F:Type, C:Type, M:Type]

operations:

estimate: $\text{Interpolator} \times \text{Observations}[F,C,M] \times C \rightarrow M$

Interpolator is a generic interface for interpolation methods. As it is an interface to other concrete types, it has no constructor. The **estimate** function takes an interpolator, an observation set and a position in space or time, and calculates a value of the measured attribute (M) for that position.

4.4 SpatioTemporal

type **SpatioTemporal**

operations:

observations: $\text{SpatioTemporal} \rightarrow \text{Observations}$

interpolator: $\text{SpatioTemporal} \rightarrow \text{Interpolator}$

begins, ends: $\text{SpatioTemporal} \rightarrow \text{Instant}$

```

boundary: SpatioTemporal → Geometry
after, before, during: SpatioTemporal x Time → SpatioTemporal
intersection, difference: SpatioTemporal x Geometry → {st1,...,stn}
                        | st: SpatioTemporal

```

axioms:

```

st1,st2: SpatioTemporal; t: Time; g: Geometry;
before(st1,begins(st1)) = Null
after(st1,ends(st1)) = Null
during(before(st1,t),t) = Null
during(after(st1,t),t) = Null
after(before(st1,t),t) = Null
before(after(st1,t),t) = Null
difference(st1,boundary(st1)) = ∅
intersection(st1,boundary(st1)) = {st1}
within(boundary(st1),g) = TRUE ⇒ intersection(st1,g)= {st1}
disjoint(boundary(st1),g) = TRUE ⇒ intersection(st1,g)= ∅
st2 ∈ intersection(st1,g) ⇒ difference(st2,g)= ∅
st2 ∈ intersection(st1,g) ⇒ boundary(st2)= g

```

The `SpatioTemporal` type provides an abstract interface to the concrete types *time series*, *trajectory*, and *coverage*. These concrete types implement the `SpatioTemporal` operations according to their needs. This type is an abstract interface and has no instances.

`Observations` and `interpolator` return the two building elements of a `SpatioTemporal` type. `Begins` and `ends` return its initial and final times. `Boundary` reports its spatial extent. `After`, `before` and `during` return a subset of a `SpatioTemporal` instance, whose temporal range is after, before and during a given time. `Intersection` and `difference` select subsets of a `SpatioTemporal` instance, whose geometries intersect and do not intersect, respectively, a given geometry.

4.5 Time Series

```

type TimeSeries [G:Geometry, T:Time, V:Value] inherits SpatioTemporal

```

operations:

```
new: Period x Observations[G,T,V] x Interpolator[G,T,V]
    → TimeSeries
value: TimeSeries x T → V
min, max: TimeSeries → V
less, greater, equals: TimeSeries x V → {ts1,...,tsn}
    | ts: TimeSeries
```

axioms:

```
ts1,ts2: TimeSeries; t1,tn: Time; v: Value;
p: Period; obs: Observations; interp: Interpolator;
ts1= new(p,obs,interp) ⇒ begins(ts1) = begin(p)
ts1= new(p,obs,interp) ⇒ ends(ts1) = end(p)
value(ts1,t1) = estimate(interpolator(ts1),observations(ts1),t1)
after(t1,ends(ts1)) v before(t1,begins(ts1)) ⇒ value(ts1,t1)=Null
value(after(ts1,t1),t1) = Null
value(before(ts1,t1),t1) = Null
less(ts1,min(ts1)) = ∅
greater(ts1,max(ts1)) = ∅
ts2 ∈ equals(ts1,v) ⇒ min(ts2) = max(ts2) = v
ts2 ∈ less(ts1,v) ⇒ max(ts2) < v
ts2 ∈ greater(ts1,v) ⇒ min(ts2) > v
boundary(ts1) = reference(observations(ts1))
positions(observations(ts1))={t1,...,tn} ⇒ begins(ts1) ≤ t1
positions(observations(ts1))={t1,...,tn} ⇒ ends(ts1) ≥ tn
```

TimeSeries is parameterized by Geometry (G), Time (T) and Value (V) types. New builds a TimeSeries from a temporal range (Period), an observation set and an interpolator. These observations have a fixed geometry (G) and measured values (V) at controlled times (T). The interpolator estimates values (V) at times during the temporal range of the series. Value uses the interpolator to provide a value at a given time. If this given time is outside the temporal range, value returns Null. Min and max return its minimum and maximum values. Less, greater and equal select subsets of a time

series whose values are, respectively, less than, greater than or equal to a given value. It inherits and implements the `SpatioTemporal` operations. For example, `boundary` returns the fixed geometry of its observations.

The temperature measures of Figure 2(a) can be represented by an `Observations[Point, Instant, Float]` type. The station location (`Point`) is fixed and the temperature (`Float`) is measured at controlled times (`Instant`). We can build a `TimeSeries[Point, Instant, Float]` from these observations. The egg traps of Figure 2(b) map to `Observations[Point, Period, Integer]`. The trap location (`Point`) is fixed and the number of eggs (`Integer`) is measured at controlled times (`Period`). We can capture the variation of the eggs in the egg traps as a `TimeSeries[Point, Period, Integer]`.

4.6 Trajectory

type `Trajectory [V:Value, T:Time, G:Geometry]` inherits `SpatioTemporal operations`:

```
new:   Period x Observations[V,T,G] x Interpolator[V,T,G]
       → Trajectory
value: Trajectory x T → G
```

axioms:

```
tj: Trajectory; t1,tn: Time; g: Geometry;
p: Period; obs: Observations; interp: Interpolator;
tj= new(p,obs,interp) ⇒ begins(tj) = begin(p)
tj= new(p,obs,interp) ⇒ ends(tj) = end(p)
value(tj,t1)= estimate(interpolator(tj),observations(tj),t1)
after(t1,ends(tj)) ∨ before(t1,begins(tj)) ⇒ value(tj,t1)=Null
value(after(tj,t1),t1) = Null
value(before(tj,t1),t1) = Null
positions(observations(tj)) = {t1,...,tn} ⇒ begins(tj) ≤ t1
positions(observations(tj)) = {t1,...,tn} ⇒ ends(tj) ≥ tn
measure(observations(tj),tn) = g ⇒ within(g,boundary(tj))=TRUE
```

Trajectory is parameterized by Value (V), Time (T) and Geometry (G) types. New constructs a Trajectory from a temporal range, an observation set and an interpolator. Trajectory observations have a fixed identity (V) and measured geometries (G) at controlled times (T). Value uses the interpolator to provide a geometry at a given time. When this given time is out of the Trajectory temporal range, value returns Null. It inherits SpatioTemporal operations and implements them according to its needs. For example, boundary returns a bounding box that contains all measured geometries of a trajectory.

Each sea elephant of Figure 3(a) is described as an instance of Observations[Integer, Instant, Point]. The animal's identity (Integer) is fixed and its location (Point) is measured at controlled times (Instant). We can capture this data as an instance of Trajectory[Integer, Instant, Point].

Each city of Figure 3(b) is described by an Observations[String, Period, MultiPolygon], where each observation contains the city's identity (String) and a boundary (MultiPolygon) valid during a period. From these observations, we build an instance of a Trajectory[String, Period, MultiPolygon] which captures the variation of a city's boundary. During the temporal range 2001 and 2012, each city's trajectory has two observations, one valid for period [2001, 2004] and the other for period [2005, 2012].

We now compare our Trajectory type with previous models such as ISO (2008) and Güting et al. (2000). Trajectory allows geometry deformations over time, whereas the ISO *moving feature model* does not (ISO 2008). Therefore, our model can cope with applications where entities change their shape, like oil spills and boundary changes in cities. The *moving point* and *moving region* defined by Güting et al. (2000) uses a linear interpolator. As Trajectory is built from an observation set and an interpolator, we can choose the most suitable interpolation function.

4.7 Coverage and Coverage Series

type

Coverage [T:Time, G:Geometry, V:Value] inherits SpatioTemporal

operations:

new: Geometry x Observations[T,G,V] x Interpolator[T,G,V]
→ Coverage

value: Coverage x G → V

min, max: Coverage → V

less, greater, equals: Coverage x V → Coverage

axioms:

cv₁,cv₂: Coverage; g: Geometry; v: Value; obs: Observations;

interp: Interpolator; t: Time;

cv₁= new(g,obs,interp) ⇒ boundary(cv₁) = g

begins(cv₁)= begin(reference(observations(cv₁)))

ends(cv₁)= end(reference(observations(cv₁)))

value(cv₁,g) = estimate(interpolator(cv₁),observations(cv₁),g)

disjoint(g,boundary(cv₁))=TRUE ⇒ value(cv₁,g) = Null

less(cv₁,min(cv₁)) = Null

greater(cv₁,max(cv₁)) = Null

equals(cv₁,v)=cv₂ ⇒ min(cv₂)= max(cv₂)= v

less(cv₁,v)=cv₂ ⇒ max(cv₂)<v

greater(cv₁,v)=cv₂ ⇒ min(cv₂)>v

less(equals(cv₁,v),v) = Null

greater(equals(cv₁,v),v) = Null

cv₂ ∈ intersection(cv₁,g) ⇒ boundary(cv₂)= g

cv₂ ∈ difference(cv₁,g) ⇒ boundary(cv₂)= boundary(cv₁)

Coverage is parameterized by Time (T), Geometry (G) and Value (V). New builds a Coverage from three elements: (1) a geometry that defines the coverage spatial extent or boundary; (2) an observation set that has a fixed time and measured values at controlled geometries; and (3) an interpolator. In most cases, the boundary is a

Polygon. However, the boundary can be other geometry types. For moving cars in a highway, the boundary could be a `MultiLineString`.

`Value` provides a value at a given location, using the interpolator. If the location is outside the coverage boundary, `value` returns `Null`. `Min` and `max` return the minimum and maximum values. `Less`, `greater` and `equal` select the coverage observations whose values are less than, greater than or equal to a given value. They return a new coverage built on such selected observations. `Coverage` inherits and implements `SpatioTemporal` operations. For example, `boundary` returns the coverage's spatial extent.

type **CoverageSeries** [G:Geometry, T:Time, CV:Coverage] inherits SpatioTemporal

operations:

new: Period x Observations[G,T,CV] x Interpolator[G,T,CV]
 → CoverageSeries

snapshot: CoverageSeries x T → CV

timeseries: CoverageSeries x Point → TimeSeries

axioms:

cs: CoverageSeries; c: Coverage; t₁,t_n: Time; l: Point;

obs: Observations; interp: Interpolator; p: Period;

cs = new(p,obs,interp) ⇒ begins(cs)= begin(p)

cs = new(p,obs,interp) ⇒ ends(cs)= end(p)

snapshot(cs,t₁) = estimate(interpolator(cs),observations(cs),t₁)

snapshot(after(cs,t₁),t₁) = Null

snapshot(before(cs,t₁),t₁) = Null

after(t₁,ends(cs)) ∨ before(t₁,begins(cs)) ⇒ snapshot(cs,t₁)= Null

begins(timeseries(cs,l))= begins(cs)

ends(timeseries(cs,l))= ends(cs)

boundary(cs) = reference(observations(cs))

measure(observations(cs),t₁)= c ⇒ boundary(cs) = boundary(c)

measure(observations(cs),t₁)= c ⇒ begins(c) = begin(t₁)

measure(observations(cs),t₁)= c ⇒ ends(c) = end(t₁)

$$\text{positions}(\text{observations}(cs)) = \{t_1, \dots, t_n\} \Rightarrow \text{begins}(cs) \leq t_1$$

$$\text{positions}(\text{observations}(cs)) = \{t_1, \dots, t_n\} \Rightarrow \text{ends}(cs) \geq t_n$$

`CoverageSeries` is an auxiliary type that represents a time-ordered set of coverages that have the same boundary. This type is useful in many applications. It is parameterized by `Geometry (G)`, `Time (T)` and `Coverage (CV)` types. Taking coverages as measured units, we construct a `CoverageSeries` from: (1) a temporal range (`Period`); (2) an observation set that has a fixed boundary (`G`) and measured coverages (`CV`) at controlled times (`T`); and (3) an interpolator that estimates coverages at non-observed times. `Snapshot` uses the interpolator to provide a coverage at a given time. If this given time is out the coverage series temporal range, `snapshot` returns `Null`. `Timeseries` returns a time series associated to a given location within the coverage series boundary.

Consider the hourly observations of air pollutions of Figure 1 obtained by cars moving in the city during one day. We can capture all observations from the same hour as an instance of `Observations[Period, Point, Float]`. These observations have a fixed time (`Period`) with measured air pollution values (`Float`) at controlled locations (`Point`). There are 24 instances of `Observations`, each leading to a `Coverage[Period, Point, Float]`. These coverages can be grouped in a `CoverageSeries[Polygon, Period, Coverage]`, producing an hourly coverage set of air pollution in the city in one day. In the rain grids of Figure 4, all observations of the same grid are represented as an instance of `Observations[Period, Point, Float]`. These observations have a fixed time (`Period`) and rain values (`Float`) at controlled cell locations (`Point`). We encapsulate each instance of `Observations` as a `Coverage[Period, Point, Float]`. Then, we group all coverages from 11 January 2011 as an instance of `CoverageSeries[Polygon, Period, Coverage]`.

Our `Coverage` type is consistent with existing field or coverage definitions (Goodchild 1992; Cova et al. 2002; OGC 2006; Liu et al. 2008). Regularly and irregularly spaced sample points can be represented by `Coverage[Point, Value, Polygon]` and isolines by `Coverage[Line, Value, Polygon]`. We can also specialize

Coverage for tessellation structures, such as raster and TIN. OGC coverage with spatiotemporal domains can be mapped to our CoverageSeries type.

4.8 Additional Functions

The proposed signatures for TimeSeries, Trajectory, Coverage and CoverageSeries types provide minimal interfaces. From those functions, a user can build more complex ones. In this section, we give some examples.

`min, max, mean, sum, mult: TimeSeries x Chronon → TimeSeries`

These operations aggregate time series values considering a given temporal resolution (Chronon) and return a new time series.

`distance: Trajectory x Trajectory → TimeSeries`

`enters, exits, reaches, leaves: Trajectory x Geometry → {tj1, ..., tjn}`
`| tji = Trajectory`

`speed: Trajectory → TimeSeries`

`direction: Trajectory → TimeSeries`

Distance computes a time series with the distance between two trajectories. Enters, exits, reaches and leaves select subsets of a trajectory that enter, exit, reach or leave a given geometry. They are based on the spatial relations between the geometries of a trajectory and a given geometry. Speed and direction return the velocity and direction variation over time.

`min, max: CoverageSeries → TimeSeries`

Min and max aggregate values of a coverage series and return a time series. We compute each value of the returned time series by taking the minimum and maximum value of a coverage at a specific time.

4.9 Object

```
type Object [ID:Value, TS:TimeSeries, TJ:Trajectory]
```

```
operations:
```

```
new: ID x TS x TJ → Object
```

```
id: Object → ID
```

```
timeseries: Object → TS
```

```
trajectory: Object → TJ
```

```
state: Object x Time → (Value, Geometry)
```

```
axioms:
```

```
o:Object; t:Time; v:Value; g:Geometry;
```

```
id(o) = reference(observations(trajectory(o)))
```

```
intersects(boundary(trajectory(o)), boundary(timeseries(o)))= TRUE
```

```
begins(trajectory(o)) = begins(timeseries(o))
```

```
ends(trajectory(o)) = ends(timeseries(o))
```

```
state(o,t) = (value(timeseries(o),t), value(trajectory(o),t))
```

An object is an identifiable entity whose spatial and non-spatial properties can change. The Object type is parameterized by its identity type (ID), a TimeSeries (TS) that represents the variation of its non-spatial property and a Trajectory (TJ) that describes the change of its spatial property. An object can have one or more non-spatial properties, but we consider only one in the type definition for simplicity. New constructs an Object. Id, timeseries and trajectory access the object parts. State returns the state of an object at a given time, that is, the values of its spatial and non-spatial properties at that time.

Each car of Figure 1 maps to an Object [Integer, TimeSeries[Polygon, Period, Float], Trajectory[Integer, Instant, Point]]. Each car's identity is represented by an Integer, its air pollution measures by a TimeSeries and its location change by a Trajectory. Each sea elephant of Figure 3 maps to an Object[Integer, \emptyset , Trajectory[Integer, Instant, Point]], where its identity is represented by an Integer and its location variation by a Trajectory. Since the sea elephants do not have non-spatial properties, they have no associated time series. Each city of the state of Rio de Janeiro in Figure 4 maps to an Object[String, TimeSeries[Polygon,

Instant, Float], Trajectory[String, Period, Polygon]]. The city name is its identity (String), the average rain variation is a TimeSeries and its boundary variation is a Trajectory. In this case, the Trajectory has a single geometry.

4.10 Event

type Event [ID:Value, T:Time, G:Geometry]

operations:

new: ID x T x G x {obj₁, obj₂, ..., obj_n} → Event

| obj: Object and $n \geq 0$

id: Event → ID

time: Event → T

location: Event → G

objects: Event → {obj₁, obj₂, ..., obj_n}

axioms:

e:Event; o:Object; t:Time; v:Value; g:Geometry;

$o \in \text{objects}(e) \wedge \text{time}(e) = t \Rightarrow \text{state}(o,t) \neq \text{Null}$

$o \in \text{objects}(e) \wedge \text{location}(e) = g$

$\Rightarrow \text{intersects}(\text{boundary}(\text{trajectory}(o)), g) = \text{TRUE}$

An event is an individual episode with a definite beginning and end which can involve one or more objects. Event is parameterized by the types of its identity (ID), time (T) and spatial location (G). New constructs an event from an identity, a time of occurrence, a geometry that stands for the event's location, and the objects involved in the event. The events of flood, dengue epidemic and animal meeting described in section 3.2 can be mapped to instances of Event[Integer, Period, Polygon]. Each instance has the event's identity (Integer), when it occurred (Period) and the region where they happened (Polygon). These events involve objects. The flood event is associated to the city of Rio. The dengue epidemic happened in the city of Recife. The meeting event involves two sea elephants.

Using operations over sets of events, we can answer questions like “*how many meetings did animal a_1 participate and where did they occur?*”, “*what meetings occurred near island x ?*”, “*when and in which districts did dengue epidemics occur in*

Recife?”, “*which are all events that occurred in Rio?*” and “*what floods have occurred in Rio during the last 5 years and what have been their average rains?*”.

Galton (2004) distinguishes punctual (instantaneous) events from durative ones (those that take time). The `Event` type can be used to represent both instances of punctual events (using `Instant`) and durative ones (using `Period`). Events associated to moving objects, such as those discussed by Hornsby et al. (2007), can also be expressed using `Event`.

5 Model Validation and Example

We tested and validated our algebra using a C++ open source geospatial software library called TerraLib (Câmara et al. 2008). Each type and its operations were implemented as *classes* and their *methods*. We also created classes to represent sets, such as `TimeSeriesSet` and `ObjectSet`, and used R-tree and B-tree for indexing geometries and times.

This section presents code examples, using the following conventions. The statement “`Type instance(p1,p2,...,pn)`” builds instance of a type using a set of parameters “`p1,p2,...,pn`”. This is equivalent to the new constructor. The code “`Trajectory a1_tj(a1_obs,interp)`” creates a `Trajectory` instance “`a1_tj`” with parameters “`a1_obs`” and “`interp`”. An operation whose first parameter is the instance and the other parameters are “`p1,p2,...,pn`” is “`instance.operation(p1,p2,...,pn)`”. This is the same as “`operation(instance,p1,p2,...,pn)`”. For example, “`a1_tj.distance(a2_tj)`” gives the distance of “`a1_tj`” and “`a2_tj`”. The command “`for each element in set {...}`” executes the commands between brackets “`{...}`” for each “`element`” of a “`set`”.

Figure 7(a) shows the code to create events of “meeting of two animals” that occur when “*the distance between two sea elephants is less than 2 meters*”. We create two trajectories “`a1_tj`” and “`a2_tj`” from observation sets “`a1_obs`” and “`a2_obs`” and interpolator “`interp`”. These are trajectories of sea elephants “`a1`” and “`a2`”, read from a KML file whose metadata is described by a XML file called “`tracks.xml`”, as described in Ferreira et al. (2012). Using “`distance`” between “`a1_tj`” and “`a2_tj`”,

returns the time series “dist”. The function “less” selects the subsets of “dist” whose values are less than 2 meters, yielding the set of time series “tsSet”. Each time series “ts” of “tsSet” leads to an event. From each “ts”, we create an event “ev” with the time (“m_per”) and place (“m_region”) of a meeting between sea elephants “a1” and “a2”.

Figure 7(b) shows the code to create events of “flood” in Rio, using the grids described in Figure 4. A ‘flood’ event occurs if *“rain is more than 10 mm/hour for more than 5 hours”*. The coverage series “cs” is built from these grids using function “createCS”, based on a metadata file “metadata.xml” and an interpolator “interp”. To select the part of “cs” inside Rio, we use the operation “intersection” that returns a coverage series “rioCS” whose boundary is the limits of Rio “rioLim”. We use operation “max” over “rioCS” to get the time series “rain”. It maps times to maximum precipitation values in Rio. Since the rain grids are taken at 15-minute intervals, the time series “rain” also contains values at each 15 minutes. Aggregating “rain” by adding the precipitation values per hour, using the operation “add” and chronon “Hour”, gets a time series “rainPerHour”. Then, we select parts of “rainPerHour” whose values are more than 10 mm/hour, using “greater”, getting a new time series set “tsSet”. Each flood event “ev” is created from a time series “ts” of “tsSet” whose extent is greater than 5 hours. All events are associated to object “rio”.

<pre> Observations a1_obs("tracks.xml", "a1"); Observations a2_obs("tracks.xml", "a2"); NearestSpaceInTimeInterp interp; Trajectory a1_tj(a1_obs, interp); Trajectory a2_tj(a2_obs, interp); Object a1("a1", a1_tj); Object a2("a2", a2_tj); ObjectSet objs; objs.add(a1); objs.add(a2); TimeSeries dist= a1_tj.distance(a2_tj); TimeSeriesSet tsSet = dist.less(2); for each ts in tsSet { Period m_per(ts.begins(), ts.ends()); Trajectory m_tj= a1_tj.during(m_per); Polygon m_region = m_tj.boundary(); Event ev("id", m_per, m_region, objs); PrintEvent(ev); } </pre> <p style="text-align: center;">(a)</p>	<pre> NearestCoverageInTimeInterp interp; CoverageSeries cs; cs = createCS("metadata.xml", interp); Polygon angraLim(...); CoverageSeries angraCS; angraCS = cs.intersection(angraLim); TimeSeries rain = angraCS.max(); Object angra("Angra", rain); TimeSeries rainPerHour = rain.max("HOURL"); TimeSeriesSet tsSet= rainPerHour.greater(10); for each ts in tsSet { if(ts.ends() - ts.begins() > 5) { Period m_per(ts.begins(), ts.ends()); Event ev("ev", m_per, angraLim, angra); PrintEvent(ev); } } </pre> <p style="text-align: center;">(b)</p>
--	--

Figure 7. Code to create events of: (a) “meeting of two animals” and (b) “flood“.

6 Final Remarks

This paper presents an algebra for spatiotemporal data types. We capture the inherent structure of geospatial observations using three types, *time series*, *trajectory* and *coverage*. Based on these types, the algebra allows defining *objects* and *events*. The proposed data types and functions can model and capture changes in a large range of applications, including location-based services, environmental monitoring, public health, and natural disasters.

A limitation of our model is to consider only two dimensional space. Since OGC geometry types can be built using 3-dimensional coordinates (x , y and z), we intend to solve this limitation in future works. In its current version, the algebra does have types that express relationships between objects and events or between events and events. These kinds of relationships, as defined by Worboys et al. (2004) and Galton et al.

(2005), can be built on top of our model. We intend to extend our algebra to represent these relationships, such as “event e_3 is composed of events e_1 and e_2 ” and “event e_1 initiates event e_2 ”.

We tested the algebra using the TerraLib software library. However, we chose to implement it in a general-purpose library that can access spatiotemporal data from different sources, including databases, files and web services. The next step is to develop an interface with the R software for statistical analysis. This includes a mapping from our types to the ones proposed by Pebesma (2011) to handle spatiotemporal data in R structures.

References

- Allen J F 1983 Maintaining Knowledge about Temporal Intervals. *Communications of the ACM* 26(11): 832-843
- Câmara G, Vinhas L, Ferreira K, Queiroz G, Souza R C M, Monteiro A M, Carvalho M T, Casanova M A and Freitas U M 2008. TerraLib: An open-source GIS library for large-scale environmental and socio-economic applications. In Hall B and Leahy M (eds) *Open Source Approaches to Spatial Data Handling*. Berlin, Springer: 247-270
- Cova T J and Goodchild M F 2002 Extending geographical representation to include fields of spatial objects. *International Journal of Geographical Information Science* 16(6): 509-532
- Ferreira K R, Vinhas L, Monteiro A M V and Camara G 2012. *Moving objects and KML files. 28th International Conference on Data Engineering (ICDE 2012) Workshop on Spatio Temporal data Integration and Retrieval*. Washington D.C., USA
- Frank A 1999 One Step up the Abstraction Ladder: Combining Algebras - From Functional Pieces to a Whole. In Freksa C and Mark D (eds.) *COSIT - Conference on Spatial Information Theory*, Springer-Verlag: 95-108
- Frank A and Kuhn W 1995. Specifying Open GIS with functional languages. In Egenhofer M and Herring J (eds) *Advances in Spatial Databases—4th International Symposium (SSD '95), Portland, ME*. Berlin, Springer-Verlag. 951: 184-195
- Galton A 2004 Fields and objects in space, time, and space-time. *Spatial Cognition and Computation* 1: 39--68
- Galton A 2008 Experience and History: Processes and their Relation to Events. *Journal of Logic and Computation* 18(3): 323-340

- Galton A and Mizoguchi R 2009 The water falls but the waterfall does not fall: New perspectives on objects, processes and events. *Applied Ontology* 4(2): 71-107
- Galton A and Worboys M 2005. *Processes and events in dynamic geo-networks. GeoSpatial Semantics (GeoS 2005)*. Rodriguez M A, Cruz I F, Levashkin S and Egenhofer M J. New York, LNCS. 3799: 45–59
- Goodchild M 1992 Geographical Data Modeling. *Computers and Geosciences* 18(4): 401-408
- Güting R H, Böhlen M H, Erwig M, Jensen C S, Lorentzos N A, Schneider M and Vazirgiannis M 2000 A Foundation for Representing and Querying Moving Objects. *ACM Transactions of Database Systems* 25(1)
- Guttag J and Horning J 1978 The Algebraic Specification of Abstract Data Types. *Acta Informatica* 10: 27-52
- Hornsby K and Egenhofer M 2000 Identity-based change: A foundation for spatiotemporal knowledge representation. *International Journal of Geographical Information Science* 14(3): 207-224
- Hornsby K S and Cole S 2007 Modeling moving geospatial objects from an event-based perspective. *Transactions in GIS* 11(4): 555-573
- ISO 2002. *Geographic information - Temporal schema (ISO 19108)*. Geneva, Switzerland, International Standard Organization
- ISO 2008. *Geographic information - Schema for moving features (ISO 19141)*. Geneva, Switzerland, International Standards Organization
- Kuhn W 2005 Geospatial Semantics: Why, of What, and How? *Journal of Data Semantics* 3: 1-24
- Kuhn W 2009 A Functional ontology of observation and measurement. In Krzysztof Janowicz M R a S L (ed. *International Conference on GeoSpatial Semantics (GeoS 2009)* Mexico City, Mexico, Springer, Lecture Notes in Computer Science
- Liu Y, Goodchild M F, Guo Q, Tian Y and Wu L 2008 Towards a general field model and its order in GIS. *International Journal of Geographical Information Science* 22(6): 623-643
- Mennis J 2010 Multidimensional map algebra: Design and implementation of a spatiotemporal GIS processing language. *Transactions in GIS* 14(1): 1-21
- OGC 2006. *OpenGIS abstract specification topic 6: Schema for coverage geometry and functions.*, Open Geospatial Consortium
- OGC 2006. *OpenGIS implementation specification for geographic information - Simple feature access - Part 1: common architecture*. Boston, Open GIS Consortium: 95
- Pebesma E 2011. *Classes and methods for spatio-temporal data in R: the spacetime package.* Available from <http://cran.r->

project.org/web/packages/spacetime/vignettes/spacetime.pdf.
Germany, Institute for Geoinformatics, University of Munster

Munster,

- Peuquet D J and Duan N 1995 An event-based spatiotemporal data model (ESTDM) for temporal analysis of geographical data. *International Journal of Geographical Information Science* 9(1): 7-24
- Regis L, Souza W V, Furtado A F, Fonseca C D, Silveira J C, Ribeiro P J, Melo-Santos M A V, Carvalho M S and Monteiro A M 2009 An entomological surveillance system based on open spatial Information for participative Dengue control. *Anais da Academia Brasileira de Ciências* 81: 655-662
- Sinton D 1978. The Inherent structure of information as a constraint to analysis: Mapped thematic data as a case study. In Dutton G (ed) *Harvard Papers on Geographic Information Systems*. Reading, MA, Addison-Wesley. 7: 1-17
- Worboys M 1994 A Unified Model for Spatial and Temporal Information. *The Computer Journal* 37(1): 27-34
- Worboys M 2005 Event-oriented approaches to geographic phenomena. *International Journal of Geographical Information Science* 19(1): 1-28
- Worboys M F and Hornsby K 2004 From objects to events: GEM, the geospatial event model. In Egenhofer M, Freska C and Miller H (eds.) *Third International Conference on GIScience*, Berlin Heidelberg New York, Springer-Verlag: 327-343
- Yuan M 1999 Three-domain representation to enhance GIS support for complex spatiotemporal queries. *Transaction in GIS* 3(2): 137-159