



Ministério da
**Ciência, Tecnologia
e Inovação**



sid.inpe.br/mtc-m19/2012/12.11.16.43-TDI

AN ALGEBRA FOR SPATIOTEMPORAL DATA: FROM OBSERVATIONS TO EVENTS

Karine Reis Ferreira

Doctorate Thesis at Graduate
Course in Applied Computing
Science, advised by Drs. Gilberto
Câmara, and Antônio Miguel
Vieira Monteiro, approved in 2012,
November, 28

URL of the original document:

<<http://urlib.net/8JMKD3MGP7W/3D76MUS>>

INPE
São José dos Campos
2012

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE

Gabinete do Diretor (GB)

Serviço de Informação e Documentação (SID)

Caixa Postal 515 - CEP 12.245-970

São José dos Campos - SP - Brasil

Tel.:(012) 3208-6923/6921

Fax: (012) 3208-6919

E-mail: pubtc@sid.inpe.br

BOARD OF PUBLISHING AND PRESERVATION OF INPE INTELLECTUAL PRODUCTION (RE/DIR-204):**Chairperson:**

Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)

Members:

Dr. Antonio Fernando Bertachini de Almeida Prado - Coordenação Engenharia e Tecnologia Espacial (ETE)

Dr^a Inez Staciarini Batista - Coordenação Ciências Espaciais e Atmosféricas (CEA)

Dr. Gerald Jean Francis Banon - Coordenação Observação da Terra (OBT)

Dr. Germano de Souza Kienbaum - Centro de Tecnologias Especiais (CTE)

Dr. Manoel Alonso Gan - Centro de Previsão de Tempo e Estudos Climáticos (CPT)

Dr^a Maria do Carmo de Andrade Nono - Conselho de Pós-Graduação

Dr. Plínio Carlos Alvalá - Centro de Ciência do Sistema Terrestre (CST)

DIGITAL LIBRARY:

Dr. Gerald Jean Francis Banon - Coordenação de Observação da Terra (OBT)

DOCUMENT REVIEW:

Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)

Yolanda Ribeiro da Silva Souza - Serviço de Informação e Documentação (SID)

ELECTRONIC EDITING:

Vivéca Sant´Ana Lemos - Serviço de Informação e Documentação (SID)



Ministério da
**Ciência, Tecnologia
e Inovação**



sid.inpe.br/mtc-m19/2012/12.11.16.43-TDI

AN ALGEBRA FOR SPATIOTEMPORAL DATA: FROM OBSERVATIONS TO EVENTS

Karine Reis Ferreira

Doctorate Thesis at Graduate
Course in Applied Computing
Science, advised by Drs. Gilberto
Câmara, and Antônio Miguel
Vieira Monteiro, approved in 2012,
November, 28

URL of the original document:

<<http://urlib.net/8JMKD3MGP7W/3D76MUS>>

INPE
São José dos Campos
2012

Cataloging in Publication Data

F413a Ferreira, Karine Reis.
An algebra for spatiotemporal data: from observations to events / Karine Reis Ferreira. – São José dos Campos : INPE, 2012.
xviii + 102 p. ; (sid.inpe.br/mtc-m19/2012/12.11.16.43-TDI)

Thesis (Doctorate Thesis at Graduate Course in Applied Computing Science) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2012.

advisors : Drs. Gilberto Câmara, and Antônio Miguel Vieira Monteiro.

1. spatiotemporal data. 2. algebra. 3. data model. 4. Geographic Information Systems. I.Título.

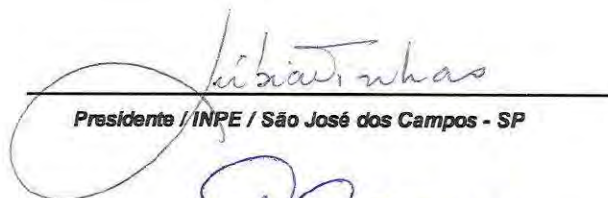
CDU 004.652

Copyright © 2012 do MCT/INPE. Nenhuma parte desta publicação pode ser reproduzida, armazenada em um sistema de recuperação, ou transmitida sob qualquer forma ou por qualquer meio, eletrônico, mecânico, fotográfico, reprográfico, de microfilmagem ou outros, sem a permissão escrita do INPE, com exceção de qualquer material fornecido especificamente com o propósito de ser entrado e executado num sistema computacional, para o uso exclusivo do leitor da obra.

Copyright © 2012 by MCT/INPE. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, microfilming, or otherwise, without written permission from INPE, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use of the reader of the work.

Aprovado (a) pela Banca Examinadora
em cumprimento ao requisito exigido para
obtenção do Título de **Doutor(a)** em
Computação Aplicada

Dra. Lúbia Vinhas



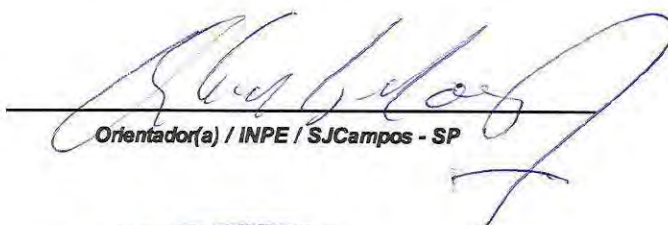
Presidente / INPE / São José dos Campos - SP

Dr. Gilberto Câmara



Orientador(a) / INPE / SJC Campos - SP

Dr. Antônio Miguel Vieira Monteiro



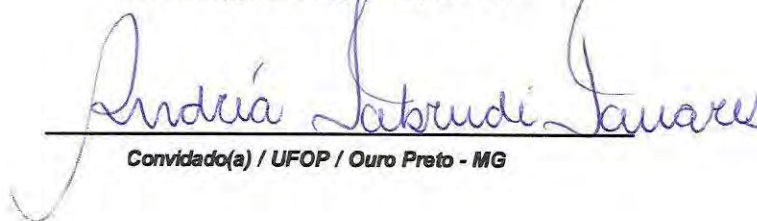
Orientador(a) / INPE / SJC Campos - SP

Dra. Vânia Bogorny



Convidado(a) / UFSC / Florianópolis - SC

Dra. Andréa Iabrudi Tavares



Convidado(a) / UFOP / Ouro Preto - MG

Este trabalho foi aprovado por:

maioria simples

unanimidade

Aluno (a): **Karine Reis Ferreira**

São José dos Campos, 28 de Novembro de 2012

*To my son Guilherme, my husband Luciano and
my parents Élcio and Terezinha.*

AGRADECIMENTOS

Muitas pessoas colaboraram com a realização desta tese e cada uma de uma maneira diferente. Algumas pessoas colaboraram *intelectualmente*. Estas me ajudaram na construção e aperfeiçoamento das ideias apresentadas nesta tese, através de discussões, sugestões e críticas. Outras me ajudaram *emocionalmente*, me apoiando com palavras e gestos de carinho nos momentos de ansiedade e cansaço que inevitavelmente fazem parte desse processo. E têm ainda as pessoas que me ajudaram a conciliar este trabalho com a maternidade, me auxiliando em momentos que tive que me ausentar para me dedicar a esta tese.

Cada ajuda teve seu valor. Agradecerei pessoalmente com um forte abraço a cada uma dessas pessoas. Lembrarei-me sempre de cada apoio!

ABSTRACT

Recent technological advances in geospatial data gathering have created massive data sets with better spatial and temporal resolution than ever. These large data sets have motivated a challenge for Geoinformatics. We need models that represent spatiotemporal data sets from different areas and that lead to 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*. Events are individual happenings with definite beginnings and ends. As a contribution to this research, this thesis proposes a model for spatiotemporal data, using an algebraic specification. Algebra gives formal specifications at a high-level abstraction, independently of programming languages. This helps to develop interoperable, reliable and expressive applications. The presented algebra is extensible, specifying data types as building blocks for other types. Three data types are defined as abstractions built on observations: *time series*, *trajectory*, and *coverage*. Using these types, we can construct *objects* and *events*. The algebra represents events explicitly, besides objects and fields. The proposed data types and functions can model and capture changes in many areas, including location-based services, public health, and environmental and natural disaster monitoring.

UMA ÁLGEBRA PARA DADOS ESPAÇO-TEMPORAIS: DE OBSERVAÇÕES A EVENTOS

RESUMO

Recentes avanços tecnológicos na aquisição de dados geográficos têm gerado uma grande quantidade de informação com melhores resoluções espaciais e temporais do que nunca. Esse grande conjunto de dados espaço-temporais tem motivado um desafio para a Geoinformática. Precisamos de modelos que representem dados espaço-temporais vindos de diferentes áreas e que auxiliem no desenvolvimento de aplicativos de boa qualidade. Muitos modelos de dados espaço-temporais existentes representam como *objetos* e *campos* evoluem ao longo do tempo. Porém, para realmente capturar mudanças, é necessário também descrever *eventos*. Eventos são acontecimentos individuais com um definitivo início e fim. Como uma contribuição para essa área de pesquisa, esta tese propõe um modelo para dados espaço-temporais, usando uma especificação algébrica. Álgebra fornece especificações formais em um alto nível de abstração, independentemente de linguagens de programação. Isto auxilia no desenvolvimento de aplicações interoperáveis, confiáveis e expressivas. A álgebra apresentada é extensível, especificando tipos de dados como unidades de construção para outros tipos. Três tipos de dados são definidos como abstrações construídas sobre observações: *time series*, *trajectory* e *coverage*. Usando esses tipos, nós podemos construir *object* e *event*. Os tipos e funções propostas podem ser usadas para modelar e capturar mudanças em uma grande variedade de aplicações, incluindo serviços baseados em localização, saúde pública e monitoramento ambiental e de desastres naturais.

LIST OF FIGURES

	<u>Page</u>
Figure 3.1 – The six possible combinations of measuring the world proposed by Sinton (1978).....	12
Figure 3.2 – Different views on observations produced by moving cars.	14
Figure 3.3 – 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).	15
Figure 3.4 – Trajectories of ten sea elephants in Antarctica (red lines).....	16
Figure 3.5 – Trajectories of three cities of Rondônia. Left and right picture present their boundaries in 2001 and 2005. Legend: blue polygon is “Costa Marques” municipality; yellow is “São Francisco do Guaporé” and green is “Seringueiras”.	16
Figure 3.6 – Example of coverages: rain in the state of Rio de Janeiro, Brazil, in 11 January 2011.	17
Figure 3.7 – Example of coverages: variation of chlorophyll in a lake of the Amazon rainforest.....	18
Figure 3.8 – Observations of a moving car and different kinds of interpolation functions.	19
Figure 3.9 – Events of meningitis in Belo Horizonte city.	20
Figure 3.10 – The proposed data model.	21
Figure 4.1 – Data types as building blocks.	24
Figure 5.1 – Software architecture: Implementation of the algebra.	43
Figure 5.2 – A simplified UML class diagram of the ST module.....	44

Figure 5.3 – Ten sea elephant trajectories loaded by the STLoader module from a KML file and the distance between two of them.....	47
Figure 5.4 – Meetings of two sea elephants.	50
Figure 5.5 – Spatiotemporal cluster of at least 3 animals.	53

LIST OF SYMBOLS

- \Rightarrow Implication. $A \Rightarrow B$ means if A is true then B is also true. If A is false then nothing is said about B .
- \vee Logical disjunction. The statement $A \vee B$ is true if A or B (or both) are true; if both are false, the statement is false.
- \wedge Logical conjunction. The statement $A \wedge B$ is true if A and B are both true; else it is false.
- $=$ Equality. The expression $x = y$ means x and y represent the same thing or value.
- \neq Inequality. The expression $x \neq y$ means that x and y do not represent the same thing or value.
- $<$ Is less than. The expression $x < y$ means x is less than y .
- $>$ Is greater than. The expression $x > y$ means x is greater than y .
- \leq Is less than or equal to. The expression $x \leq y$ means x is less than or equal to y .
- \geq Is greater than or equal to. The expression $x \geq y$ means x is greater than or equal to y .
- $\{ \}$ Set. $\{a, b, c\}$ means the set consisting of a , b , and c .
- $|$ Such that. $S = \{(x, y) \mid 0 < y < f(x)\}$ means the set of (x, y) such that y is greater than 0 and less than $f(x)$.
- \in Is an element of. The expression $a \in S$ means a is an element of the set S .
- \emptyset Empty set. It means the set with no elements.

SUMMARY

	<u>Page.</u>
1 INTRODUCTION.....	1
1.1. The Proposal.....	2
1.2. Document Structure.....	3
2 RELATED WORK.....	5
2.1. Changes in Objects.....	5
2.1.1. Moving Objects and Trajectories.....	6
2.2. Changes in Fields.....	7
2.3. Events.....	7
2.3.1. Continuants and Occurrents.....	8
2.3.2. Event Representation.....	8
2.4. Our Approach.....	9
3 FOUNDATIONS.....	11
3.1. Observations.....	11
3.2. Data Abstractions.....	13
3.2.1. Time Series.....	14
3.2.2. Trajectory.....	15
3.2.3. Coverage.....	16
3.2.4. Interpolation Functions.....	18
3.3. Objects and Events.....	19
4 AN AGEBRA FOR SPATIOTEMPORAL DATA.....	23
4.1. Primitives Data Types.....	24

4.2. Observations	25
4.3. Interpolator	26
4.4. SpatioTemporal	26
4.4.1. Time Series	28
4.4.2. Trajectory	30
4.4.3. Coverage and Coverage Series	32
4.4.4. Additional Functions	36
4.5. Object	38
4.6. Event	40
5 PROOF OF CONCEPT AND EXAMPLES	43
5.1. Software Architecture	43
5.2. Code Examples	47
5.2.1. Meeting of Two Animals	48
5.2.2. Spatiotemporal Clusters.....	50
5.2.3. Flood	53
6 FINAL REMARKS AND FUTURE WORK	57
6.1. Next Steps Related to Software Implementation	58
6.2. Comparison with Previous Work.....	59
ANNEX A – TOWARDS A DYNAMIC GEOSPATIAL DATABASE MODEL	67
ANNEX B – MOVING OBJECTS AND SPATIAL DATA SOURCES	83

1 INTRODUCTION

The age of big geospatial data has come. Mobile phones, social networks and GPS (Global Positioning System) devices create useful data for planning better cities, capturing human interactions and improving life quality. 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. We now have large spatiotemporal data sets in many application domains. These massive data sets have motivated a challenge for GIScience. We need models that represent spatiotemporal data sets from different areas and that lead to good quality software.

In GIScience, static geospatial information is represented following well-established models and concepts. This includes the dichotomy between *object-based* and *field-based* models. *Objects* are identifiable geographical units with spatial and non-spatial attributes and *fields* are mapping from spatial locations to values of a property (COUCLELIS, 1992; GOODCHILD, 1992; WORBOYS and DUCKHAM, 2004). Examples of long-standing concepts are *vector* and *raster* data structures, *topological operators*, *spatial indexing*, and *spatial joins* (EGENHOFER; FRANZOSA, 1991; RIGAUX et al., 2002). Most existing geographical information systems (GIS) and spatial database systems are grounded in these concepts. However, there is no consensus on how to represent spatiotemporal information in computational systems.

Many existing proposals of spatiotemporal data models focus on representing the evolution of objects and fields over time. Pelekis et al. (2004) review some of these models and consider that most of them are data-specific; each one addresses a class of spatiotemporal data. Some proposals are specific for discrete changes in objects (WORBOYS, 1994; YUAN, 1999; HORNSBY; EGENHOFER, 2000), others for moving objects (MARK et al., 1999; GÜTING et al., 2000; ISO, 2008) and still others for fields or coverage (PEUQUET; DUAN, 1995; LIU et al., 2008; OGC, 2006b; MENNIS, 2010). However, many

applications need to combine different classes of such data. For example, environmental change and natural disaster monitoring have to deal with moving objects as well as with fields. Thus, we need spatiotemporal data models as generic as possible to support such applications.

To properly capture changes in the world, representing evolution of objects and fields over time is not enough. We also need to represent *events* and relationships between events and objects explicitly (WORBOYS, 2005). Events are *occurrences* (GALTON; MIZOGUCHI, 2009). They are individual happenings with definite beginnings and ends. The demand for models that describe events has encouraged recent research on spatiotemporal data modeling (WORBOYS; HORNSBY, 2004; GALTON, 2004; GALTON; WORBOYS, 2005; HORNSBY; COLE, 2007; WORBOYS, 2005).

1.1. The Proposal

To meet all these demands, this thesis proposes a data model for spatiotemporal information using an algebraic specification. The main contribution of this work is an extensible algebra to represent variation of objects and fields over time as well as events. The proposed data types and functions can model and capture changes in many areas, including location-based services, public health, and environmental and natural disaster monitoring. Algebras describe data types and their operations in a formal way, independently of programming languages. By separating specification from implementation, they help to develop interoperable, reliable and expressive applications (FRANK; KUHN, 1995; FRANK, 1999).

The presented model starts with observations, revisiting the Sinton's classical work (SINTON, 1978). Recent research draws attention to the importance of using observations as a basis for designing geospatial applications. Observations are our means to assess spatiotemporal phenomena in the real world (KUHN, 2009). Kuhn (2005) argues that: "*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 proposed algebra is extensible. It defines data types as building blocks for other types. Three data types are defined as abstractions built on observations: *time series*, *trajectory*, and *coverage*. Using these types, we can construct *objects* and *events*. The algebra represents events explicitly, besides objects and fields. An event contains information about *when* and *where* it occurred and its involved objects.

Since algebraic specifications are language-independent, programmers can translate them into software using programming languages of their choice. As an example, we have tested and validated the proposed algebra by implementing its data types and functions using the C++ language. Two modules have been developed on top of the geospatial software library TerraLib (CÂMARA et al., 2008).

1.2. Document Structure

The content of this document basically comes from three papers:

- (1) Ferreira, K. R.; Câmara, G.; Monteiro, A. M. *Towards a Dynamic Geospatial Database Model*. In: The International Conference on Emerging Databases (EDB 2011), 2011, Incheon, Korea. The Third International Conference on Emerging Databases (EDB 2011). Incheon, Korea, 2011. v. 1.
- (2) Ferreira, K. R.; Vinhas, L.; Monteiro, A. M.; Câmara, G. *Moving Objects and Spatial Data Sources*. (Accepted for publication in September 2012 in the journal “Revista Brasileira de Cartografica”).

- (3) Ferreira, K. R.; Câmara, G.; Monteiro, A. M. *An Algebra for Spatiotemporal Data: From Observations To Events*. (Accepted for publication in October 2012 in the journal “Transactions in GIS”).

Chapter 2 describes some existing spatiotemporal data models and related work. It is a summarized and revised version of paper (1) together with the related work reported in paper (3).

Chapter 3 and 4 present the foundations and the algebraic specification of the proposed data model. They come from the core of paper (3), extending it with new examples of the algebra data types and with useful additional functions.

Chapter 5 describes how the proposed algebra was tested and validated and presents its use examples. This chapter merges some parts of paper (2) with some use examples of paper (3). Besides that, this chapter provides details about the algebra implementation using the geographical software library TerraLib.

Chapter 6 concludes the work. Annex A and B contain the complete papers (1) and (2), respectively. Since the core of paper (3) is entirely presented along this thesis, it has not been annexed to this document.

2 RELATED WORK

This chapter presents a review of some existing spatiotemporal data models and related work, grouping them in three categories: (1) models that represent changes in objects; (2) models that represent evolution of fields over time; and (3) models that represent events explicitly.

2.1. Changes in Objects

Some authors distinguish *instantaneous* from *continuous* changes in objects (GALTON, 2004; GÜTING and SCHNEIDER, 2005). Cases of instantaneous changes arise mostly due to legal rules that demand an immediate change in an object. When a government creates laws that alter municipality limits, changes take effect instantaneously. Continuous changes refer to a constant variation over time and space. Examples include the movement of cars on a highway and of migratory animals. In these cases, the spatial locations of cars and animals change continuously over time.

Galton (2004) points out the difference between *bona fide* and *fiat* object behavior over time. *Bona fide* objects are grounded in features of physical reality, such as rivers and forest regions. *Fiat* objects are the artificial products of human cognitive acts, such as municipality limits and land parcels. He argues that: “*Both these objects might change over time, but typically the bona fide entity will undergo gradual change whereas the fiat entity undergoes sudden change (as a result of the boundary being redrawn from time to time).*” He uses the terms “gradual” and “sudden” to refer to continuous and instantaneous changes, respectively. Güting and Schneider (2005) distinguish *discrete* from *continuous* changes and argue that classical research on spatiotemporal database has focused on discrete changes of all the spatial entities. They define *moving objects* as entities whose spatial location or boundary change continuously over time and propose a model to deal with them.

Worboys (1994), Yuan (1999) and Hornsby and Egenhofer (2000) propose models to represent instantaneous changes in objects. Worboys (1994) proposes a *unified spatiotemporal object model* that defines two data types, *ST-simplexes* and *ST-complexes*, and a set of operations over them, such as *ST-Union*, *ST-Intersection* and *ST-Difference*. Yuan (1999) defines a *three-domain model* to represent variation of objects over time in relational database systems, using normalized tables and a spatial graph. Hornsby and Egenhofer (2000) present a model for changes in identifiable objects, expressing operations like *create*, *destroy* and *continue existence*.

2.1.1. Moving Objects and Trajectories

Recent growth of mobile computing has motivated much work on moving objects. Mark et al. (1999), Güting et al. (2000) and ISO (2008) propose models that represent continuous changes in the spatial location or extent of objects. Mark et al. (1999) define the concept of *geospatial lifeline* that models an individual's movement as a time-stamped record of locations. Güting et al. (2000) define an algebra, data types and operations over them, for moving objects. Interest in location-based applications led to an ISO (2008) standard that defines a *moving feature* as an object whose geometry moves as a rigid body.

Based on the algebra proposed by Güting et al. (2000), there are two main initiatives of Moving Object Database (MOD) systems, *SECONDO* (GUTING; SCHNEIDER, 2005) and *Hermes* (PELEKIS et al., 2008). Both extend the SQL type system with data types to represent moving objects, such as *moving point* and *moving region*, and a set of functions to deal with them. *SECONDO* is an extensible database system prototype designed at the FernUniversität in Hagen. *Hermes* is a MOD engine that has been implemented as an Oracle data cartridge.

Spaccapietra et al. (2007) propose a conceptual model for trajectories of moving objects. They define trajectories as countable journeys that are

semantically segmented by defining a temporal sequence of time intervals when the object position changes (*moves*) and stays fixed (*stops*). In the data mining research area, many algorithms, techniques and languages have been proposed to detect patterns of trajectories (LAUBE et al., 2005; BOGORNY et al., 2009; SAKR; GÜTING, 2011).

2.2. Changes in Fields

To represent fields, Goodchild (1992) proposes two models: *sampling and interpolation-based field* and *tessellation-based field*. Cova and Goodchild (2002) define *object-field* as a mapping from continuous field to discrete objects. Liu et al. (2008) introduce the concept of *general field* and show how conventional fields as well as *object-fields* can be seen as specializations of it. A general field has three dimensions in space plus one dimension in time. Efforts on standardization led to the OGC Coverage Standard (OGC, 2006b). This standard uses the term *coverage* to refer to *field* and defines it as a feature that associates positions in a spatial, temporal or spatiotemporal domain to attribute values. Although its definition includes spatiotemporal domains, only coverages with spatial domains are described in its UML class diagrams. For example, the domain of *CV_DiscretePointCoverage* consists in a set of points (*GM_Point*) and of *CV_DiscreteCurveCoverage* in a set of curves (*GM_Curve*).

Raster is a particular representation of fields based on a regular cell grid. Some models are specific to represent changes in raster. Peuquet and Duan (1995) propose a model that groups changes in raster cells by time of occurrence. Mennis (2010) extends the conventional map algebra to multidimensional raster, including two or three dimensions in space and one dimension in time.

2.3. Events

Before talking about event representation, the next section introduces what event means.

2.3.1. Continuants and Occurrents

According to Galton (2008), in philosophical ontology there is a long-standing classification of real world phenomena into *continuants* and *occurrents*. Continuants (or *endurants*) are entities whose identities remain constant as they undergo change, such as an aircraft and a volcano. They are present as a whole at each moment of their existence. Occurrents are entities that happen or occur, like a flight and an eruption. They cannot undergo change and only exist as a whole across the interval over which they occur.

In the ontology literature, there is not a universal agreement about what *events* and *processes* are and how they are classified, *continuants* or *occurrents*. Some ontologies classify both as occurrents, such as SNAP/SPAN (GRENON; SMITH, 2004). Others categorize processes as continuants and events as occurrents, such as EXP/HIST (GALTON, 2008). This thesis follows the definitions and classifications proposed by Galton and Mizoguchi (2009). An event is an occurrent. It is an individual episode with a definite beginning and end. An event is a chunk of some process. A process is indefinitely extended in time. They are the “stuff” of which events are composed of.

2.3.2. Event Representation

Some spatiotemporal data models represent event explicitly. Worboys and Hornsby (2004) propose a unified model for objects and events. This model defines two kinds of relationships, *object-event* and *event-event*, following the idea that an event can affect or be associated to one or more objects or events. Some examples of object-event relationships are *splitting* and *merger* (“*the event e_1 created or destroyed the boundary between objects o_1 and o_2* ”). Some examples of event-event relationship are *initiation* and *termination* (“*the occurrence of event e_1 started or terminated the event e_2* ”). Galton and Worboys (2005) refine such relationships for events, states, and processes in dynamic networks.

Hornsby and Cole (2007) model events associated to moving objects and propose an approach to extract patterns of movements from them. An event contains its name, the identifier of the object associated to it, the spatial region where it occurred, and the instant when it happened. Worboys (2005) presents a pure event oriented model, using an algebraic approach. He argues that “*happenings should be upgraded to an equal status with things in dynamic geographic representations*” and suggests ways of doing so.

2.4. Our Approach

This thesis focuses on defining an extensible algebra that covers the whole process to obtain *events* from raw *observations*. We represent events explicitly, besides the variation of objects and fields over time. This is the main difference between our approach and the previous ones described in this chapter. In the proposed model, an event contains information about *when* and *where* it occurred and its involved objects. We do not define types of relationships between objects and events neither between events and events. These kinds of relationships, as the ones defined by Worboys and Hornsby (2004) and Galton and Worboys (2005), can be constructed on top of the presented model.

3 FOUNDATIONS

This chapter presents the base concepts on which the proposed algebra is grounded, illustrating them with real examples. It is basically part of the paper entitled “*An Algebra for Spatiotemporal Data: From Observations To Events*” that has been accepted for publication in the journal “Transactions in GIS”.

3.1. Observations

The proposed model starts with observations, which are our means to assess spatiotemporal phenomena in the real world (KUHN, 2009). It uses observations as the basis for spatiotemporal modeling, following Sinton’s approach (SINTON, 1978). According to Sinton, 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 to the 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. This means to: (1) keep one attribute constant; (2) vary the second attribute in a controlled way; and (3) measure the third attribute, taking into consideration the constraints of the second attribute. This produces six possible combinations, shown in Figure 1.

	Fixed	Controlled	Measured	Examples
(1)	Location	Time	Theme	Weather reports*
(2)	Time	Location	Theme	Grid cell data*
(3)	Theme	Time	Location	Tracks of animals
(4)	Location	Theme	Time	Arrival times by runners
(5)	Time	Theme	Location	Vegetation Map*
(6)	Theme	Location	Time	Tide Tables*

* Examples from Sinton (1978)

Figure 3.1 – The six possible combinations of measuring the world proposed by Sinton (1978).

This work proposes three data types, *time series*, *coverage* and *trajectory*, to represent the combinations (1), (2) and (3) presented in Figure 1:

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

We consider that these three data types are necessary and sufficient to model spatiotemporal data. All the six combinations presented in Figure 1 can be modeled as *time series*, *trajectory* or *coverage*. We do not need additional data types to represent the combinations (4), (5) and (6). Combination (4) occurs in cases like “measuring arrival times by runners in a marathon”. In this case, it is possible to get this type of information by analyzing *trajectories* of runners, without needing an additional data type. As an example of combination (5),

Sinton proposes a “vegetation map” that could be obtained by finding out all locations of a given land cover type. This is an awkward way to get a land cover map. Usually, such maps are obtained by a systematic data collection over a given area, resulting in *coverages*. Sinton suggests “tide tables” as an example for combination (6). Since such tables can be obtained from a *time series* that maps times to tide heights at a specific location, an additional type is needless. Thus, we consider that only three data types (*time series*, *coverage*, and *trajectory*) are necessary to model all possible combinations of theme, time, and space.

3.2. Data Abstractions

The model defines three data types as abstractions built on observations: *time series*, *trajectory*, and *coverage*. Using these types, we can create different views on the same observation set, meeting application needs. Consider a set of cars equipped with GPS and air pollution sensors. Figure 2 shows tracks of three cars in a city during one day. These cars produce an observation set, where each one contains a car identity, a time instant, a location and an air pollution value. The observations are taken at each hour.

From this data, we can extract three different kinds of information: (1) *how the average air pollution varies over time in the city*; (2) *how the cars move over time and space*; (3) *how pollution varies within the city limits*. A typical query in the case (1) is “*When the average pollution in the city was greater than x for more than five hours?*”; in the case (2) is “*How long did car c_{01} stay in the south region of the city?*”; in the case (3) is “*What city district had the worst pollution index in this day?*”. Thus, each application needs different queries and each kind of query is suited to a specific data type. Taking the whole city as a fixed reference, we can get a *time series* that represents the variation of the average air pollution in the city per hour. Considering each car an individual object, we can get a set of *trajectories*. Fixing the whole day as a time reference and taking

all observations at that day, we can create a *coverage* to represent the air pollution variation within the city limits during that day.

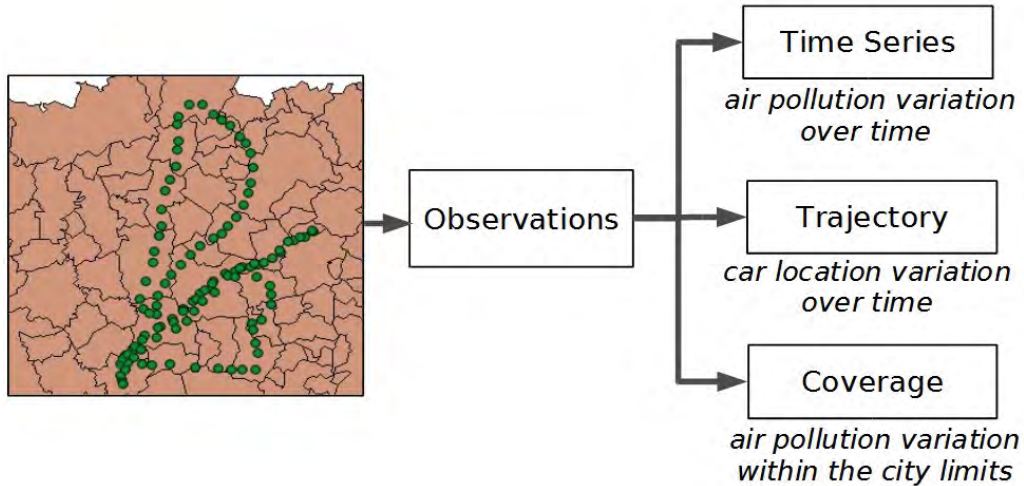


Figure 3.2 – Different views on observations produced by moving cars.

3.2.1. Time Series

A *time series* represents the variation of a property over time. It is obtained from observations that measure values at controlled times in a fixed location. Figures 3(a) and 3(b) show time series used in disease surveillance of dengue in the city of Recife, Brazil (REGIS et al., 2009). Dengue is a viral disease transmitted by the *Aedes aegypti* 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 3(a) shows five meteorological stations and one temperature time series. A second set of time series represents the number of mosquito eggs gathered weekly from egg traps. Figure 3(b) presents egg traps (red points) in a district of Recife and a time series produced by one of them.

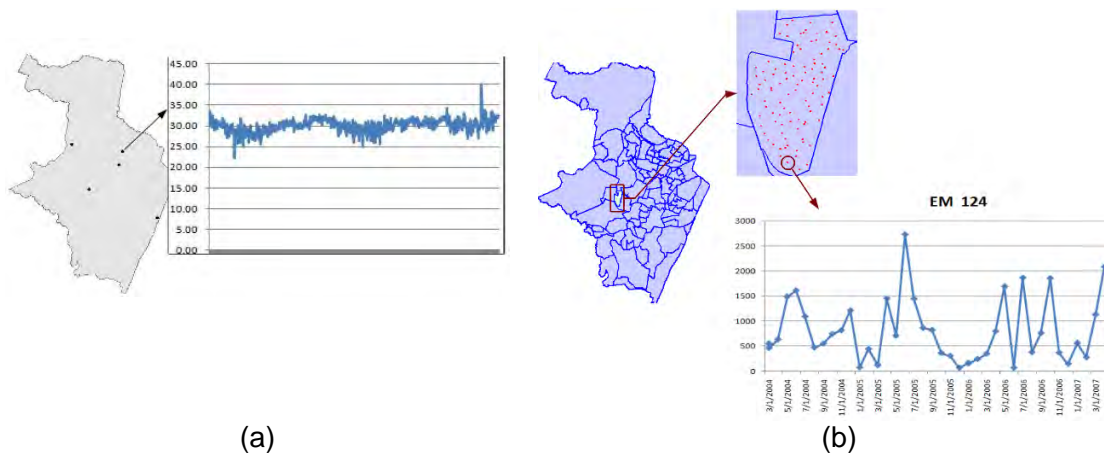


Figure 3.3 – Examples of time series: (a) temperature collected by meteorological stations and (b) number of mosquito eggs gathered from one egg trap in a district of Recife (Brazil).

3.2.2. Trajectory

A *trajectory* represents how locations or boundaries of an object change over time. Figures 4 and 5 show examples of trajectories. Figure 4 presents routes of ten sea elephants in Antarctica. These animals are monitored by a project called MEOP - “Marine Mammal Exploring the Oceans Pole to Pole” (<http://www.inpe.br/crs/pan/pesquisas/telemetria.php>). Figure 5 shows the evolution of three city limits in the Brazilian state of Rondonia from 2001 to 2005.

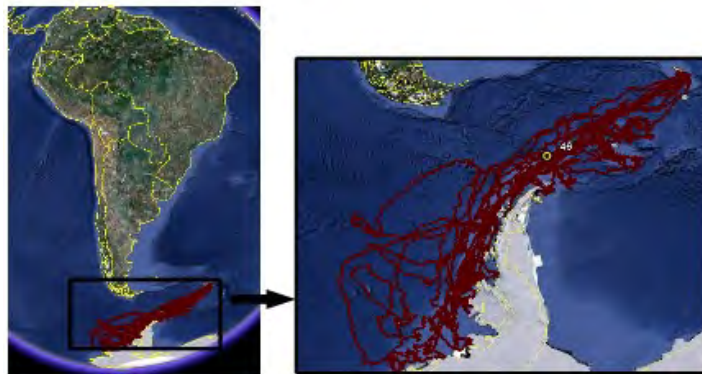


Figure 3.4 – Trajectories of ten sea elephants in Antarctica (red lines).

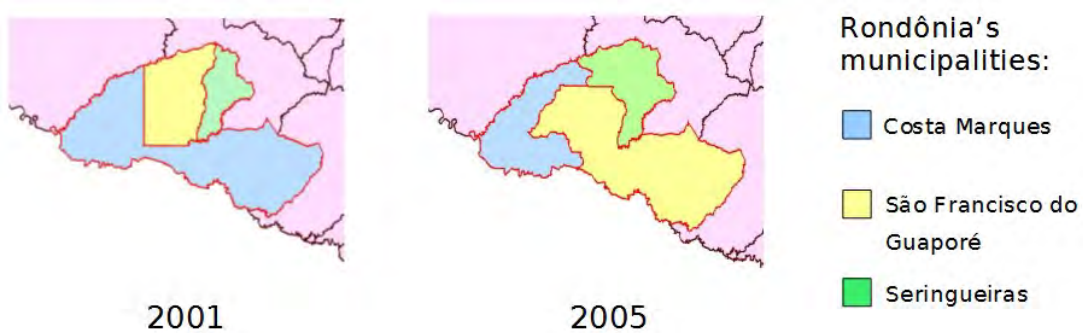


Figure 3.5 – Trajectories of three cities of Rondônia. Left and right picture present their boundaries in 2001 and 2005. Legend: blue polygon is “Costa Marques” municipality; yellow is “São Francisco do Guaporé” and green is “Seringueiras”.

3.2.3. Coverage

A *coverage* represents the variation of a property in a spatial extent at a time. For every location within such extent, it is possible to compute a value of this property. The variation of air pollution in the city districts during one day (Figure 2) is represented by a coverage that has the observations obtained by all cars. Figure 6 shows examples of coverages, where each one is represented by a grid associated to a time. Each coverage is a grid associated to a time. These grids contain the rain variation in the state of Rio de Janeiro during the natural

disaster of 11 January 2011. Each cell contains an estimated value of precipitation, in millimeter per hour (mm/h).

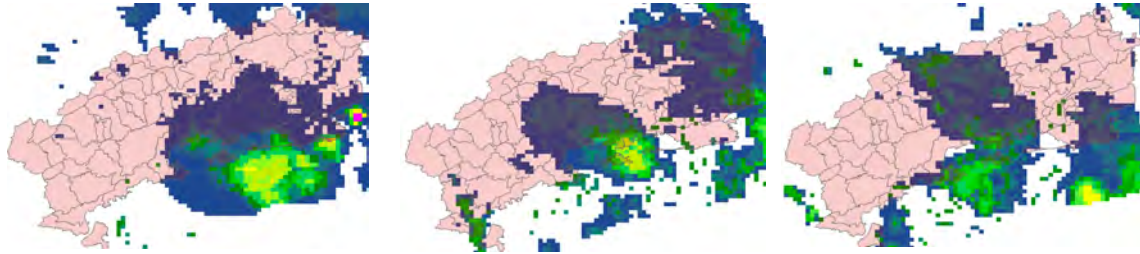


Figure 3.6 – Example of coverages: rain in the state of Rio de Janeiro, Brazil, in 11 January 2011.

A set of observations in an Amazon rainforest lake is shown as red points in Figure 7, in four different months. Each observation measures the chlorophyll value, among other properties, at a specific location and time. These observations are taken to analyze the variation of chlorophyll within the lake over time. Usually, a kriging interpolation function is used to estimate values at non-observed locations in the lake. In our model, the observations of each month are represented as a coverage whose spatial extent is the limits of the lake.

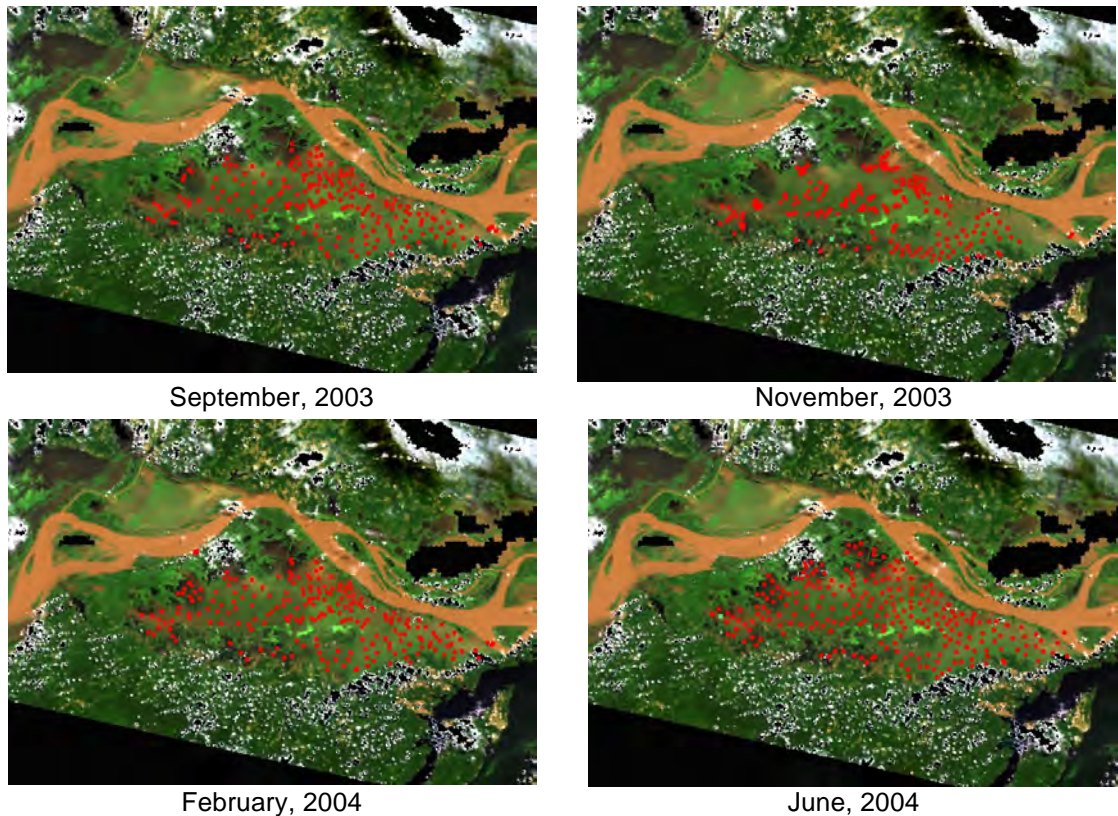


Figure 3.7 – Example of coverages: variation of chlorophyll in a lake of the Amazon rainforest.

3.2.4. Interpolation Functions

Since observations are discrete by nature, we need to combine them with interpolation functions to approximate continuous change. Interpolators estimate values at locations in space and moments in time for which there is no data (KNOTTERS et al., 2010). Consider two observations of a moving car (Figure 2), one at instant 4 and the other at 8, shown in Figure 8 (a). There are different methods to estimate car location at the non-observed time 6. Choices include a linear interpolator (Figure 8 (b)) or a method that uses a street map as a spatial constraint, as in Figure 8 (c). The proposed algebra allows a user to choose the most suitable interpolation function for each type instance.

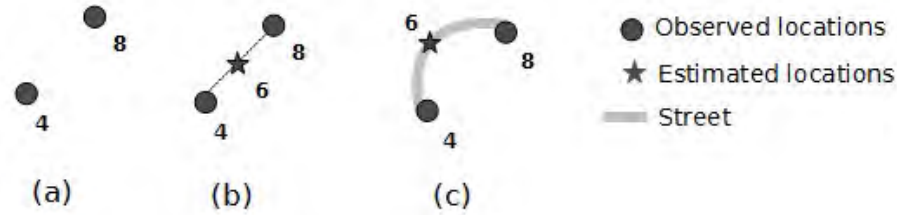


Figure 3.8 – Observations of a moving car and different kinds of interpolation functions.

3.3. Objects and Events

The model defines objects as *continuants* and events as *occurents*. 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; MIZOGUCHI, 2009). Examples of objects are cars (Figure 2), egg traps (Figure 3), sea elephants (Figure 4), cities (Figure 5) and cities of Rio de Janeiro (Figure 6). 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; MIZOGUCHI, 2009). The proposed model represents events and their involved objects explicitly. 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 them using operations over the proposed types.

Consider the following objects and conditions that lead to events. The objects are the cities of Angra dos Reis and Recife and a group of sea elephants. A ‘flood’ event occurs in Angra 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° 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’ event occurs when “*the minimal distance between two sea elephants is shorter than 2 meters*”. We can express these conditions through operations on

time series, *trajectories* and *coverages*, which in turn are built from observations.

The proposed model can also represent events that are not directly derived from conditions of objects. Since we have information about *when* and *where* an event happened, we can represent it using the model. Examples include occurrences of crimes or diseases in a city. Figure 9 presents occurrences of meningitis in Belo Horizonte city (black points). Each event has a spatial location and a time of occurrence. We can also associate each event to the district object where it occurred. Figure 10 presents an overview of the proposed model.

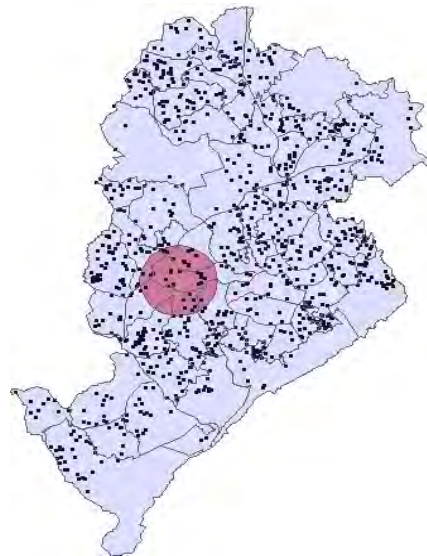


Figure 3.9 – Events of meningitis in Belo Horizonte city.

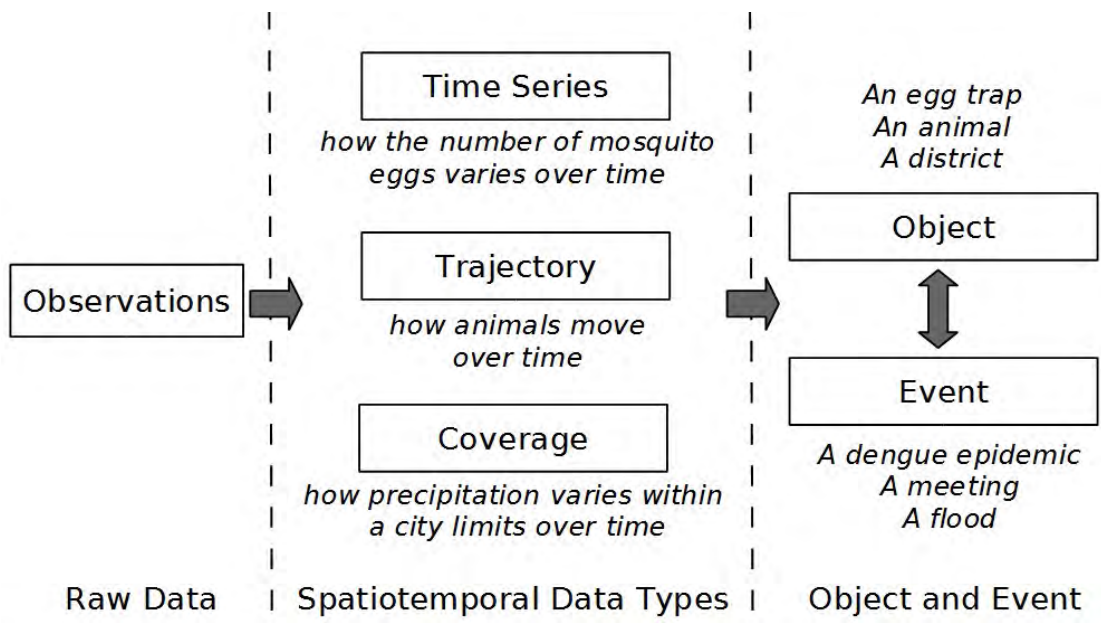


Figure 3.10 – The proposed data model.

4 AN ALGEBRA FOR SPATIOTEMPORAL DATA

This chapter presents an algebraic specification of the data model described in Chapter 3. It is basically part of the paper entitled “*An Algebra for Spatiotemporal Data: From Observations To Events*” that has been accepted for publication in the journal “Transactions in GIS”.

Data types are used to express abstractions. A *data type* is a set of values and a collection of operations on those values that defines their behavior. 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 (GUTTAG et al, 1978). In our specification, 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.

The proposed algebra defines data types as building blocks to create other types, as shown in Figure 11. It starts defining a set of primitive types: Number, Value, Time, Chronon and Geometry. Then, it specifies the Observations and Interpolator types that are used to build the spatiotemporal data types. The spatiotemporal types are TimeSeries, Trajectory, Coverage and CoverageSeries. They implement an abstract interface defined by the SpatioTemporal type. Object type is built from Trajectory and TimeSeries. Event is created from Object.

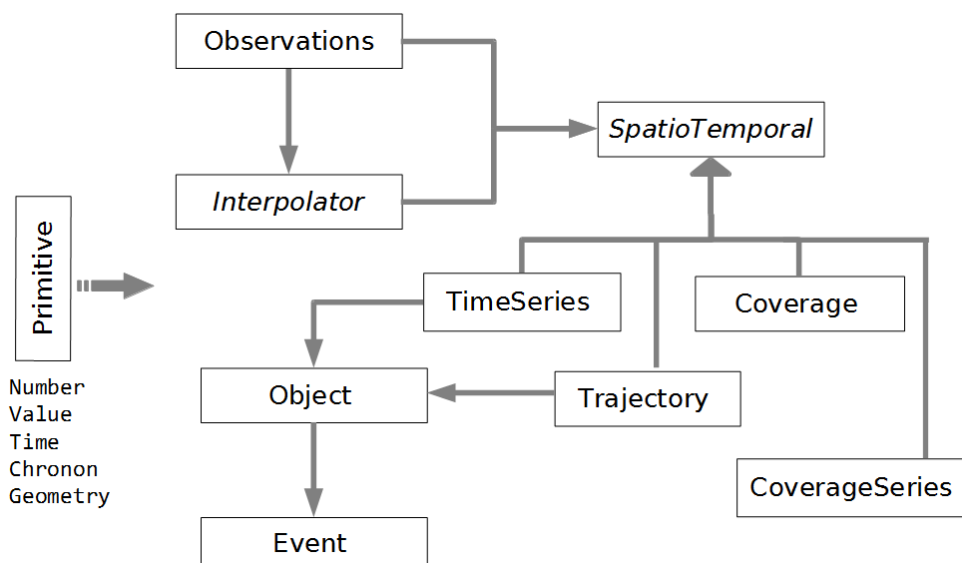


Figure 4.1 – Data types as building blocks.

4.1. Primitives Data Types

There are three primitive types: Value, Time and Geometry. Value is a generic type to express attribute values that can be an Integer, Float, String or Boolean. Typical operations on Value type include `less_than`, `greater_than`, `equal_to`, `max`, and `min`. The meanings of such operations are 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. Our types Time, Instant and Period are compliant with *TM_GeometricPrimitive*, *TM_Instant* and *TM_Period* defined in 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 (1991). Their behavior 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, 2006a). 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 (2006a). The types are:

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

These types and their operations are well-known and have already been defined in the adopted standard. 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 behavior 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 elements (F,C,M) of any types. The Observations type represents a set of observations and 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 tuples of types F, C and M. Reference returns the value of the fixed attribute. The positions function reports the controlled attribute values. Measure returns the observed value associated to a given position.

4.3. Interpolator

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

operations:

estimate: Interpolator x Observations[F,C,M] x C → M

The Interpolator type defines a generic interface for interpolation methods. A programmer will create concrete implementations of it, one for each interpolation method. It has no constructor, since it is an interface to other concrete types. The Interpolator type has only a function called estimate. This function takes an observation set and a position in space or time of the same type of the controlled attribute. Then, it calculates a value (M) valid for that position.

4.4. SpatioTemporal

type: **SpatioTemporal**

operations:

observations: SpatioTemporal → Observations

interpolator: SpatioTemporal → Interpolator

begins, ends: SpatioTemporal → Instant

boundary: SpatioTemporal \rightarrow Geometry

after, before, during: SpatioTemporal x Time

\rightarrow SpatioTemporal

intersection, difference: SpatioTemporal x Geometry

\rightarrow $\{st_1, \dots, st_n\} \mid st: \text{SpatioTemporal and } n \geq 0$

axioms:

$st_1, st_2: \text{SpatioTemporal}; t: \text{Time}; g: \text{Geometry};$

$\text{before}(st_1, \text{begins}(st_1)) = \text{Null}$

$\text{after}(st_1, \text{ends}(st_1)) = \text{Null}$

$\text{during}(\text{before}(st_1, t), t) = \text{Null}$

$\text{during}(\text{after}(st_1, t), t) = \text{Null}$

$\text{after}(\text{before}(st_1, t), t) = \text{Null}$

$\text{before}(\text{after}(st_1, t), t) = \text{Null}$

$\text{difference}(st_1, \text{boundary}(st_1)) = \emptyset$

$\text{intersection}(st_1, \text{boundary}(st_1)) = \{st_1\}$

$\text{within}(\text{boundary}(st_1), g) = \text{TRUE} \Rightarrow \text{intersection}(st_1, g) = \{st_1\}$

$\text{disjoint}(\text{boundary}(st_1), g) = \text{TRUE} \Rightarrow \text{intersection}(st_1, g) = \emptyset$

$\text{intersection}(st_1, g) = \{st_2\} \Rightarrow \text{difference}(st_2, g) = \emptyset$

$\text{intersection}(st_1, g) = \{st_2\} \Rightarrow \text{boundary}(st_2) = g$

The SpatioTemporal type provides an abstract interface to the concrete types *time series*, *trajectory*, and *coverage*. It contains common operations of these three types. These concrete types inherit SpatioTemporal operations and

implement them according to their needs, conforming to the axioms above. As this type is only a generic interface, it has no instances.

The spatiotemporal types are built from an observation set and an interpolator. The operations `observations` and `interpolator` return these two building elements. `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 a `SpatioTemporal` instance, whose geometries intersect and do not intersect, respectively, a given geometry.

4.4.1. 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` → {`ts`₁,`ts`₂,...,`ts`_n}
| `ts`: `TimeSeries` and `n` ≥ 0

axioms:

`ts`₁,`ts`₂: `TimeSeries`; `t`₁,`t`_n: `Time`; `v`:`Value`;

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

`ts`₁ = `new`(`p`,`obs`,`interp`) ⇒ `begins`(`ts`₁) = `begin`(`p`)


```

ts1 = new(p, obs, interp) ⇒ ends(ts1) = end(p)
value(ts1, t1) =
    estimate(interpolator(ts1), observations(ts1), t1)
after(t1, ends(ts1)) ∨ 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)) = ∅
equals(ts1, v) = {ts2} ⇒ min(ts2) = max(ts2) = v
less(ts1, v) = {ts2} ⇒ max(ts2) < v
greater(ts1, v) = {ts2} ⇒ 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 (T) 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. All these behaviors are described in the axioms. 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, as described in the axioms.

The temperature measures of Figure 3 (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 instants (`Instant`). We can build a `TimeSeries[Point, Instant, Float]` using these observations. The observations of the each traps of Figure 3 (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 traps as a `TimeSeries[Point, Period, Integer]`.

4.4.2. 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)) v 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. The 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. All these behaviors are described in the axioms. 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.

The observations of each sea elephant of Figure 4 is described as an instance of Observations[Integer, Instant, Point] type. The animal 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]. The observations of each city shown in Figure 5 is described by an instance of Observations[String, Period,

MultiPolygon] type, 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 the 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].

4.4.3. 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 this given location is outside the coverage boundary, Value returns Null. All these behaviors are described in the axioms. 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)

$\text{measure}(\text{observations}(cs), t_1) = c \Rightarrow \text{begins}(c) = \text{begin}(t_1)$

$\text{measure}(\text{observations}(cs), t_1) = c \Rightarrow \text{ends}(c) = \text{end}(t_1)$

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

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

`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 of 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 2 obtained by cars moving in the city during one day. We can capture all observations of 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 6, 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].

Considering the chlorophyll measurement shown in Figure 7, all observations of the same month are represented as an instance of Observations[Instant, Point, Float]. Each observation is associated to the instant when it was collected. We encapsulate each instance of Observations as a Coverage[Period, Point, Float]. Each coverage is associated to the period that represents a month. Then, we group all coverages as an instance of CoverageSeries[Polygon, Period, Coverage].

4.4.4. 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 of additional functions.

Some additional operations for TimeSeries:

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

union: TimeSeries x TimeSeries → TimeSeries

Min, max, mean, sum, and mult aggregate time series values considering a given temporal resolution (Chronon) and return a new time series. They calculate each value of the new time series by taking the minimum, maximum, average, sum or multiplication of all values in the same time resolution. Union computes the union between two given time series ts_1 and ts_2 and returns a new one ts_3 . For each time t_n of ts_1 and ts_2 , it gets the observations ob_{1n} of ts_1 and ob_{2n} of ts_2 at time t_n . If ob_{1n} exists, it uses it to create ts_3 . If ob_{1n} does not exist and ob_{2n} exists, it uses ob_{2n} to create ts_3 .

Some additional operations for Trajectory:

distance: Trajectory x Trajectory → TimeSeries

enters, exits, reaches, leaves: Trajectory x Geometry

→ {tj₁,...,tj_n} | tj: Trajectory and n≥0

speed: Trajectory → TimeSeries

direction: Trajectory → TimeSeries

linearPath: Trajectory → Line

convexhullPath: Trajectory → Polygon

necklacePath: Trajectory → PolygonSet

Distance computes the distance between two trajectories, tj_1 and tj_2 . It returns a time series that maps time to the Euclidean distance between both at that time. 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 trajectory geometry tjg and a given geometry g . If tjg is disjoint from g at t_i and within g at t_{i+1} , the trajectory enters g in period $[t_i, t_{i+1}]$. If tjg is within g at t_i and disjoint from g at t_{i+1} , it exits g in period $[t_i, t_{i+1}]$. If tjg is disjoint at t_n and touches at t_{n+1} , it reaches g in period $[t_n, t_{n+1}]$. If tjg intersects at t_n and disjoint at t_{n+1} , it leaves g in period $[t_n, t_{n+1}]$.

Speed and direction return the velocity and direction variation over time. LinearPath, convexhullPath and necklacePath return trajectory approximations, based on the ones defined by Hornsby and Egenhofer (2002). LinearPath, convexhullPath and necklacePath are specific to trajectories whose G is a Point. Therefore, when we apply them to trajectories whose geometries are not points, we consider their centroids.

Some additional operations for CoverageSeries:

min, max, mean, sum, mult: CoverageSeries → TimeSeries

min, max, mean, sum, mult: CoverageSeries x Geometry
→ TimeSeries

Min, max, mean, sum and mult aggregate values of a coverage series and return a time series. We can define a spatial restriction given a geometry, as presented in the second signature. They compute each value of the returned time series by taking the minimum, maximum, average, sum and multiplication of all values of the coverage series at a specific time. If there is a spatial restriction, they consider only the values whose locations intersect the given geometry.

4.5. 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 2 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 4 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 6 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.

Non-spatial properties of an object can be derived from coverage series. For example, the average rain variation within the city limits can be extracted from the coverage series presented in Figure 6, using CoverageSeries operations.

The next chapter presents a case study that includes this example. Using operations over objects, we can answer questions that combine the variation of spatial and non-spatial properties like “*where are all the cars now and what are the pollution indexes associated to them?*” and “*where were the cars when the pollution indexes associated to them were more than x?*”.

4.6. 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 ≥ 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. The Event type 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 3.3 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 Angra dos Reis. 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 Angra dos Reis during the last 5 years and what have been their average rains?”*.

5 PROOF OF CONCEPT AND EXAMPLES

This chapter describes how the proposed algebra was tested and validated and presents its use examples. We have implemented the algebra data types and functions, using the C++ programming language, on top of the geographical software library TerraLib (CÂMARA et al., 2008). TerraLib is an open source library base to build geographical information systems.

5.1. Software Architecture

We have developed two new modules called ST and STLoader on top of TerraLib DataAccess module, as shown in Figure 12. The ST module contains all data types and functions of the proposed algebra. Each type and its operations were implemented as *classes* and their *methods*. Figure 13 shows a simplified UML (Unified Modeling Language) class diagram of the ST module.

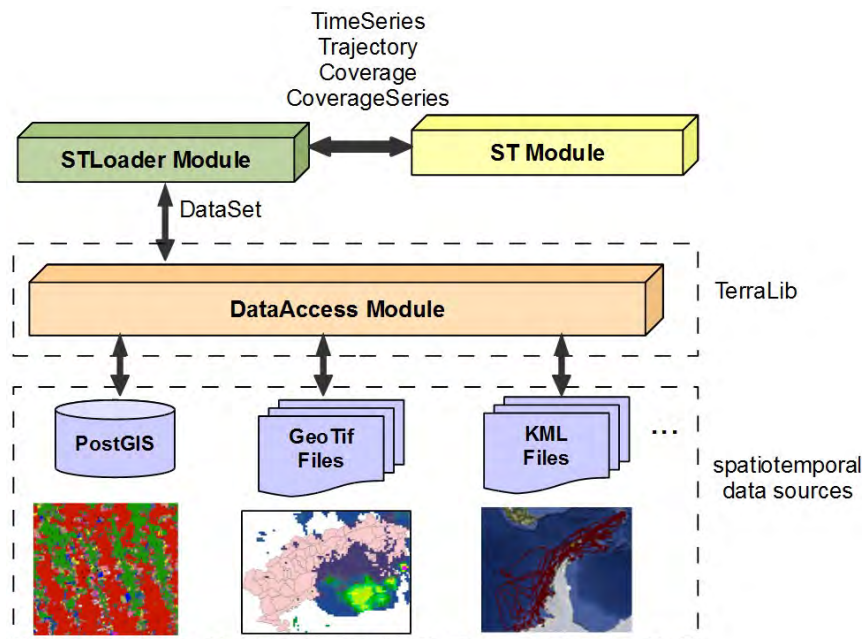


Figure 5.1 – Software architecture: Implementation of the algebra.

The classes *AbstractObservationSet*, *TimeSeries*, *Trajectory*, *Coverage*, *CoverageSeries*, *Object* and *Event* implement the algebra's types

Observations, TimeSeries, Trajectory, Coverage, CoverageSeries, Object and Event, respectively. The classes *PointCoverage* and *RasterCoverage* specialize the *Coverage* class. *PointCoverage* represents coverages whose observations are taken by measuring values at controlled locations (that is, G is a Point type). *RasterCoverage* represents coverages that use raster structures to associate locations to values. The proposed class architecture is extensible. We can extend it to other kinds of coverages. For example, we can also specialize the *Coverage* class for isolines and triangulated irregular network (TIN), creating the classes *LineCoverage* and *TINCoverage*.

Considering the examples of coverages shown in Section 3.2.3, each precipitation grid (Figure 6) is represented as an instance of *RasterCoverage* and all chlorophyll observations of the same month (Figure 7), as an instance of *PointCoverage*.

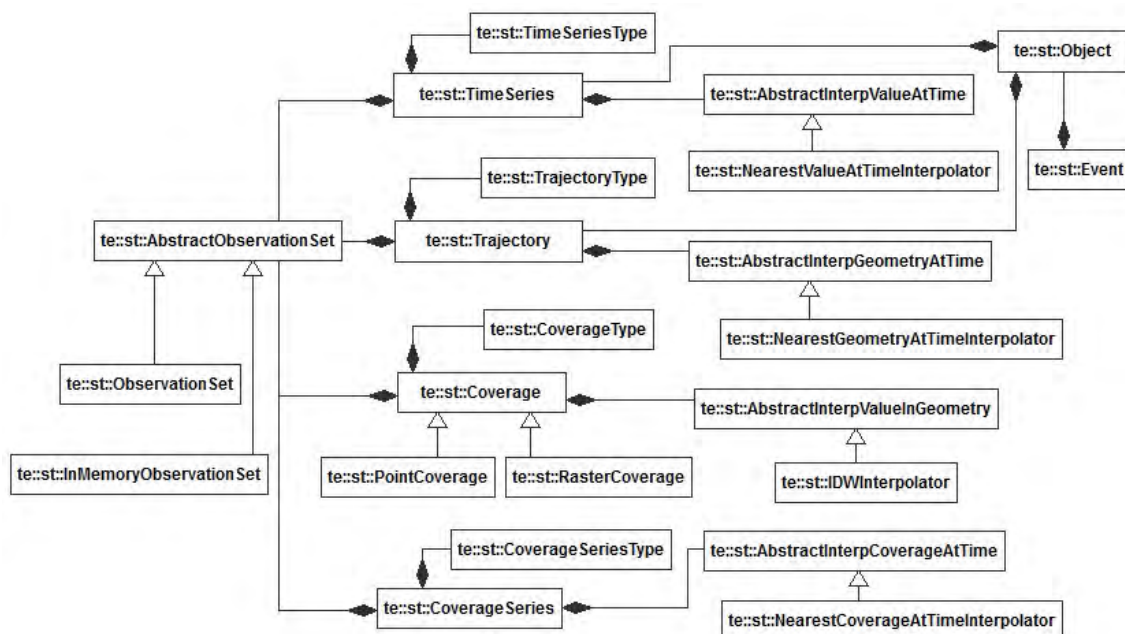


Figure 5.2 – A simplified UML class diagram of the ST module.

The abstract classes *AbstractInterpValueAtTime*, *AbstractInterpGeometryAtTime*, *AbstractInterpValueInGeometry* and *AbstractInterpCoverageAtTime* implement the Interpolator type of the algebra. All these classes have an operation called estimate that calculates values at non-observed positions. The types of the estimated value and of the position vary according to each interpolator. *AbstractInterpValueAtTime* estimates values of the type Value (Number, Boolean or String) at non-observed times. *AbstractInterpGeometryAtTime* estimates geometries at non-observed times. *AbstractInterpValueInGeometry* estimates values of the type Value at non-observed locations. *AbstractInterpCoverageAtTime* estimates coverages at non-observed times.

NearestValueAtTimeInterpolator, *NearestGeometryAtTimeInterpolator*, *IDWInterpolator* and *NearestCoverageAtTimeInterpolator* are concrete classes that implement the abstract interpolators. Given a non-observed time, the estimate function of the first class provides the closest measured value to that given time. The estimate function of the second class provides the nearest measured geometry to the given non-observed time. As an example, to estimate the car location at non-observed time 7 in Figure 8, *NearestGeometryAtTimeInterpolator* provides the measured location at time 8. When estimating the car location at time 5, it gives the measured location at time 4.

IDWInterpolator implements an Inverse Distance Weighted (IDW) interpolator. *NearestCoverageAtTimeInterpolator* provides the closest measured coverage to the given non-observed time. As an example, to estimate the coverage at non-observed time "December, 2003" in Figure 7, *NearestCoverageAtTimeInterpolator* gives the closest measured coverage, that is, the coverage at time "November, 2003". As the proposed class architecture is extensible, the idea is to improve this module by providing many other kinds of interpolators, for example kriging for coverages and street-based interpolator for trajectories.

As defined in the algebra, the classes *TimeSeries*, *Trajectory*, *Coverage* and *CoverageSeries* are composed of an observation set (*AbstractObservationSet*) and of an interpolator (*AbstractInterpValueAtTime*, *AbstractInterpGeometryAtTime*, *AbstractInterpValueInGeometry* or *AbstractInterpCoverageAtTime*). The *Object* class is composed of *TimeSeries* and *Trajectory*. The *Event* class is composed of *Object*.

The *DataAccess* module of TerraLib can load data sets with spatial and temporal information from different kinds of data sources, such as PostGIS databases as well as KML and GeoTif files. The *STLoader* module is responsible for transforming these data sets into instances of the spatiotemporal types defined in the *ST* module. To perform this transformation, we have proposed a strategy based on metadata files. The strategy and its validation using trajectories of moving objects have been reported in a paper entitled "Moving Objects and Spatial Data Sources". Annex B presents the complete paper.

The paths of ten sea elephants presented in Figure 4 are stored in a KML file as sets of spatial locations associated to time stamps. Using the proposed strategy (shown in Annex B), the *STLoader* module loads and transforms these sets into instances of the *Trajectory* class of the *ST* module. Thus, using the *Trajectory* functions we can analyze these sets as moving object trajectories. For example, we can answer questions like: (1) *Where was animal₁ at time t₅?* (2) *When did animal₁ reach the island x and how long did it stay in this island?* (3) *When and where did animal₁ and animal₂ meet each other (considering a meeting when the distance between two animals is less than 10 meters)?* (4) *Where and when was there a spatiotemporal cluster of animals?*

Figure 14 shows the ten sea elephant trajectories (colored lines at the bottom) loaded by the *STLoader* module from the KML file and the distance between two of them. As defined in the algebra, the distance operation between two trajectories results in a time series, shown in the left side of the picture.

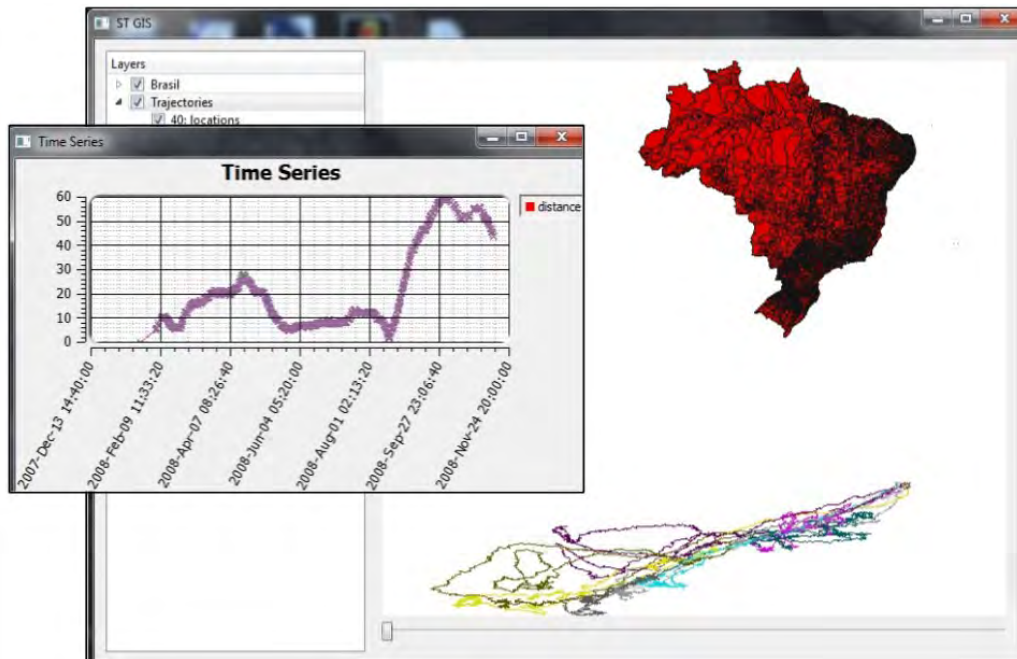


Figure 5.3 – Ten sea elephant trajectories loaded by the STLoader module from a KML file and the distance between two of them.

5.2. Code Examples

This section presents pseudo code examples of how to get events from spatiotemporal data types, using the proposed data types and functions. Using the data sets presented in Section 3, this section shows three code examples that cover all data types and a substantial set of functions. The code uses the following conventions.

The statement `Type instance(p1,p2,...,pn)` builds an instance called “instance” of the type “Type” using the set of parameters “p₁,p₂,...,p_n”. This is equivalent to the new constructor. The command `Event ev("Meeting", p, b, objs)` creates the instance “ev” of the type “Event” with four parameters, “Meeting”, “p”, “b” and “objs”. To execute an operation “operation” whose first parameter is the instance “instance” and the others are “p₁,p₂,...,p_n”, we use the command `instance.operation(p1,p2,...,pn)`. It is equivalent to `operation(instance,p1,p2,...,pn)`. For example, `tj1.distance(tj2)`

executes operation “distance” of the instance “tj1” with the parameter “tj2”. It is equivalent to “distance(tj1,tj2)”.

To represent a set of instances of a type “Type”, we use “set<Type>”. The statement “for each *element* in set {...}” executes the commands between brackets “{...}” for each “*element*” of the “set”. For example, the command “set<TimeSeries> result” creates a set called “result” of instances of “TimeSeries”. To access each instance “ts” of “result”, we use “for each ts in result”. The statement “Type inst1 = inst2.operation(p₁,p₂,...,p_n)” creates an instance called “inst1” of the type “Type” from the result of the operation “operation”. For example, “TimeSeries dist = tj1.distance(tj2)” builds the instance “dist” of the type “TimeSeries” from the operation “distance”.

5.2.1. Meeting of Two Animals

Considering two sea elephants (identifiers 43 and 44) shown in Figure 4, the following code creates events of “meeting of two animals” that occur when “*the distance between two sea elephants is less than 10 meters*”:

```
Trajectory tj1;
Trajectory tj2;
LoadTrajectories("metadata.xml", tj1, tj2);

set<Object> objs;
Object seaElephant43(43, 0, tj1);
Object seaElephant44(44, 0, tj2);
objs.add(seaElephant43);
objs.add(seaElephant44);

TimeSeries dist = tj1.distance(tj2);
set<TimeSeries> result = dist.less(10.0);
```

```

set<Event> meetings;
for each ts in result
{
    Period      p(ts.begins(), ts.ends());
    Trajectory  tj = tj1.during(p);
    Geometry    b = tj.boundary();

    Event ev("Meeting", p, b, objs);
    meetings.add(ev);
}

```

The code above creates two trajectories “tj1” and “tj2” and loads them from the KML file presented in Figure 4 through a function called “LoadTrajectories”. This function is part of the STLoader module and loads trajectories from different data sources based on metadata files, as described in Annex B. The metadata file “metadata.xml” describes the two trajectories to be loaded. We do not inform explicitly what interpolator should be used. In this case, the function “LoadTrajectories” associates the default interpolator “NearestGeometryAtTimeInterpolator” to each trajectory. Afterwards, the code creates two objects “seaElephnat43” and “seaElephnat44” whose identifiers are “43” and “44” and trajectories are “tj1” and “tj2”. These objects do not have non-spatial properties, that is, they do not have time series associated to them. We create the set of objects “objs” and add “seaElephnat43” and “seaElephnat44” to it.

To identify a meeting, we calculate the distance between “tj1” and “tj2”, using the function “distance”. It gives the time series “dist”. So, we select parts of “dist” whose values are less than 10, using the operation “less”. It returns the time series set “result”. Each time series “ts” of “result” indicates an event. From each “ts”, we get the period (“p”) and the region (“b”) where the meeting occurred. Then, we create an event “ev” using such information and its involved

objects “seaElephnat43” and “seaElephnat44”, which are in the object set “objs”. All events are added to the set “meetings”.

Figure 15 shows three events of meetings between two animals detected using the code above. The location of each event is presented as a red rectangle at the bottom of the picture. This picture also presents the distance between the two animals as a time series. Observing this time series, we can identify when these events occurred. There are three parts of it whose values are less than 10.

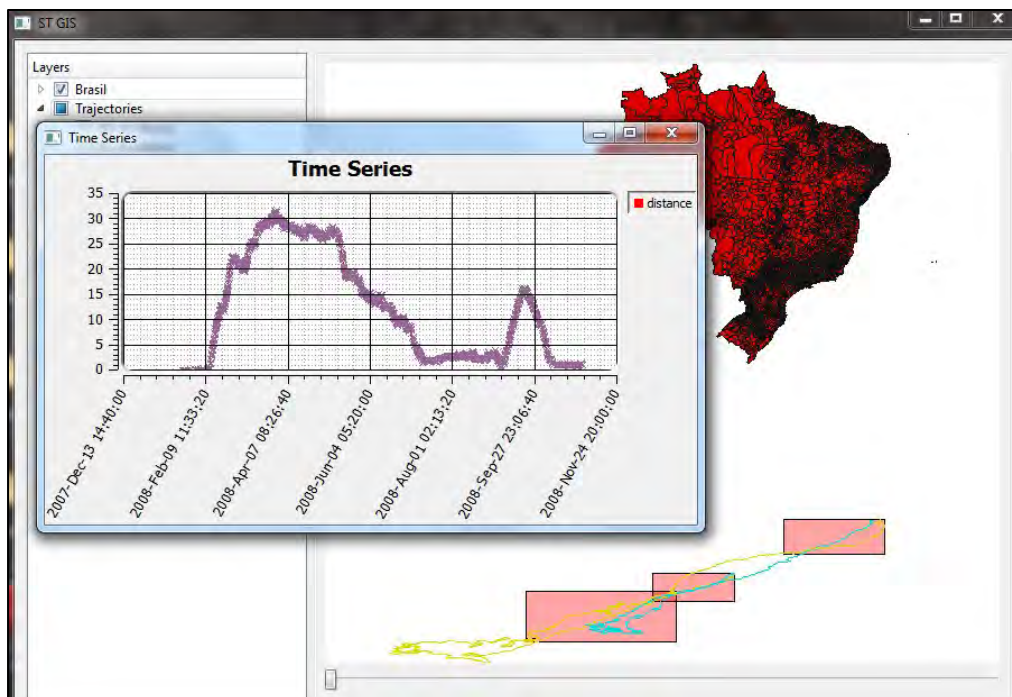


Figure 5.4 – Meetings of two sea elephants.

5.2.2. Spatiotemporal Clusters

Considering five sea elephant paths presented in Figure 4, the following code creates events of “spatiotemporal cluster of animals”. We consider that a spatiotemporal cluster occurs when “the sum of the distances among the animals (at least three) is less than 10 meters for at least 15 consecutive days”:

```

set<Trajectory> paths;
LoadTrajectories("metadata5.xml", paths);

set<Object> objs;
Object seaElephant40(40, 0, paths[0]);
Object seaElephant41(41, 0, paths[1]);
Object seaElephant42(42, 0, paths[2]);
Object seaElephant43(43, 0, paths[3]);
Object seaElephant44(44, 0, paths[4]);
objs.add(seaElephant40);
objs.add(seaElephant41);
objs.add(seaElephant42);
objs.add(seaElephant43);
objs.add(seaElephant44);

TimeSeries* distancePerDay = MinDistancePerDay(paths);
set<TimeSeries> result = distancePerDay.less(10.0);

set<Event> clusters;
for each ts in result
{
    if(ts.ends() - ts.begins() > 15)
    {
        Period      p(ts.begins(), ts.ends());
        Geometry    l = GetLocation(paths, p);
        Event ev("ST Cluster", p, l, objs);
        clusters.add(ev);
    }
}

```

The code above creates a set of trajectories “paths” and loads the trajectories of five sea elephants from the KML file presented in Figure 4 through a function called “LoadTrajectories”. This function is part of the STLoader module and

loads trajectories from different data sources based on a metadata file, as described in Annex B. In this example, the metadata file “metadata5.xml” describes the five sea elephants to be loaded. We do not inform explicitly what interpolator should be used. In this case, the function “LoadTrajectories” associates the default interpolator “NearestGeometryAtTimeInterpolator” to each trajectory. Afterwards, the code creates five objects “seaElephnat40”, “seaElephnat41”, “seaElephnat42”, “seaElephnat43”, “seaElephnat44” and “seaElephnat45” from the loaded set of trajectories “paths”. These objects do not have non-spatial properties, that is, they do not have time series associated to them. We create the set of objects “objs” and add these objects to it.

To identify spatiotemporal clusters, the function “MinDistancePerDay” calculates a time series that contains the sum of the three minimal distances among the five animals per day. To create the time series “distancePerDay”, this function calculates the distances among all five trajectories using the operations “distance” and selects only the three minimal distances per day. So, we select parts of “distancePerDay” whose values are less than 10, using the operation “less”. It returns the time series set “result”. Each time series “ts” of “result” whose temporal range is greater than 15 days ($ts.ends() - ts.begins() > 15$) indicates an event. The period of each event is created from the temporal range (“p”) of “ts”. The event location (“l”) is computed through the function “GetLocation” that gives the region of the trajectories “paths” associated to period “p”. All events are added to the set “clusters”.

Using the code above, we have found one event of spatiotemporal cluster. Figure 16 shows the location of this event as a red rectangle at the bottom of the picture. This figure also presents the time series “distancePerDay” calculated in the code above.

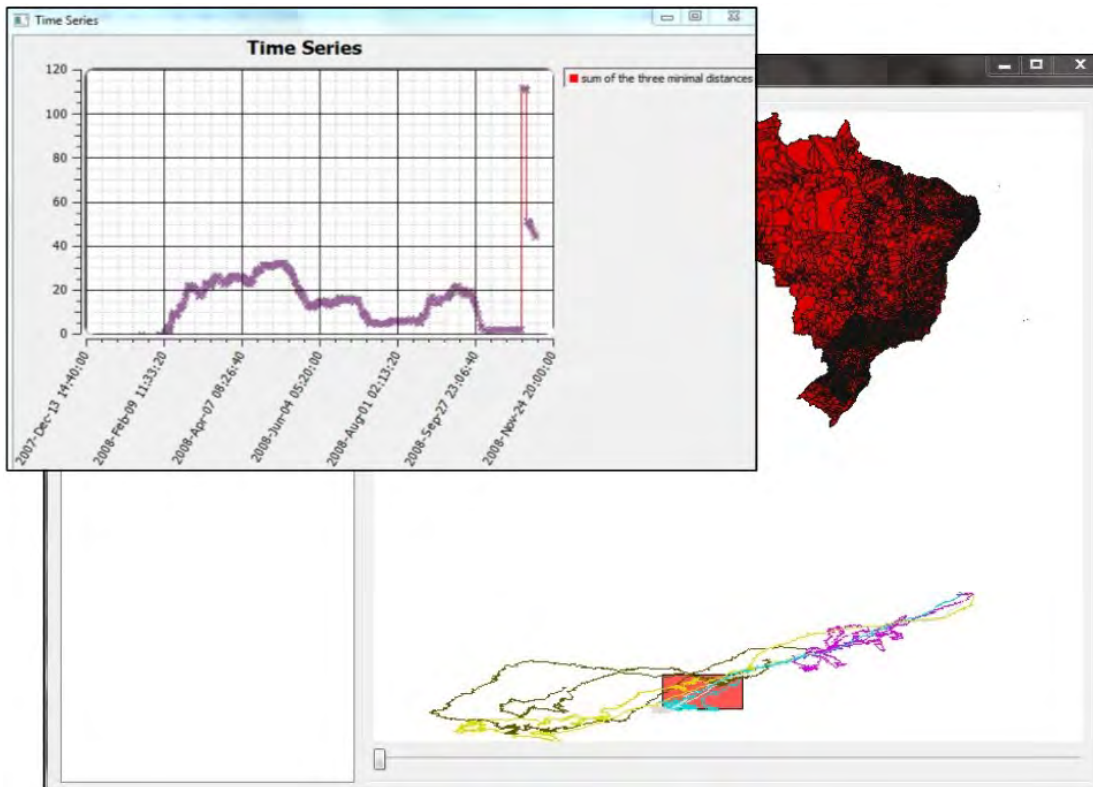


Figure 5.5 – Spatiotemporal cluster of at least 3 animals.

5.2.3. Flood

Using the rain grids presented in Figure 6, the following code creates events of “flood” in Angra city that occur if “rain is more than 10 mm/hour for more than 5 hours”:

```
CoverageSeries cvseries;
LoadCoverageSeries("metadata.xml", cvseries);

DataSet cities;
LoadDataSet("postgis&...&br_cities", cities);
Polygon angraLimits = cities.getGeometry("Angra dos Reis");

TimeSeries maxRainInAngra = cvseries.max(angraLimits);
TimeSeries maxRainInAngraPerHour = maxRainInAngra.max(HOUR);
```

```

set<TimeSeries> result = maxRainInAngraPerHour.greater(10);

Object angraCity("Angra dos Reis", maxRainInAngraPerHour, 0);
set<Event> floods;
for each ts in result
{
    if(ts.ends() - ts.begins() > 5)
    {
        Period time(ts.begins(), ts.ends());
        Event ev("Flood", time, angraLimits, angraCity);
        floods.add(ev);
    }
}

```

The code above creates a coverage series “cvseries” and loads it from the precipitation grids shown in Figure 6 through a function called “LoadCoverageSeries”. Each grid associated to a time is a geotif file. This function is part of the STLoader module and loads coverage series from different data sources based on the metadata file “metadata.xml”, following the approach described in Annex B. In this case, the metadata file contains the name and path of each geotif file and the time associated to it. Internally, the function “LoadCoverageSeries” creates an instance of the RasterCoverage type for each grid. Since we do not inform explicitly what interpolator should be used, this function associates the default interpolator “NearestCoverageAtTimeInterpolator” to the coverage series.

The boundary of Angra city called “angraLimits” is get from the data set “cities” loaded from a PostGIS database through the function “LoadDataSet”. We use the operation “max” and the spatial restriction “angraLimits” to get the time series “maxRainInAngra”. This time series maps times to the maximum rain in Angra city. Since the rain grids are taken at 15-minute intervals, the time series “maxRainInAngra” also contains values at each 15 minutes. So, we

aggregate “maxRainInAngra” by taking the maximum precipitation values per hour, using the operation “max” and chronon “Hour”. This returns the new time series “maxRainInAngraPerHour”. Then, we select parts of “maxRainInAngraPerHour” whose values are more than 10 mm/hour, using the operation “greater”. This provides the new time series set “result”.

Each event of flood “ev” is created from a time series “ts” of “result” whose temporal range is greater than 5 hours (“if(ts.ends() - ts.begins() > 5)”). All events are associated to the object “angraCity” and are added to the set “floods”.

6 FINAL REMARKS AND FUTURE WORK

This thesis presents an extensible algebra for spatiotemporal data. Taking observations as basic building blocks, the algebra constructs three data types, *time series*, *trajectory* and *coverage*. This allows us to define different views on the same observation set, meeting application needs. Considering coverages as measured units, we extend the algebra defining the *coverage series* type. Using these types, we can define *objects* and *events*. The proposed data types and functions can model and capture changes in a large range of applications, including location-based services, public health, and environmental and natural disaster monitoring.

The proposed model addresses both *instantaneous* and *continuous* changes in objects, as discussed in Section 2.1. Paths of animals (Figure 4) as well as changes of municipality limits (Figure 5) can be represented as instances of Trajectory type. The boundary variation of a city is represented by a set of observations. Each observation contains its valid boundary in a period and the periods of all observations cover the whole timeline of the city. Therefore, for every time during this timeline, there is a valid boundary of the city and the interpolator always provides it.

A limitation of the proposed 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 work. The Event type contains information about *when* and *where* it occurred as well as its involved objects. However, the algebra does not express *how* events are related to other events. These kinds of relationships, as defined by Worboys and Hornsby (2004) and Galton and Worboys (2005), can be built on top of our model. We intend to extend the model 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 also plan to define functions to create types from other types. For example, we can create a set of *TimeSeries* from the *CoverageSeries* composed of

instances of *RasterCoverage* presented in Figure 6. In this case, each time series will be associated to a cell or pixel and will contain the variation of rain in that cell over time. We can create a *CoverageSeries* composed of instances of *PointCoverage* from the *TimeSeries* shown in Figure 3 (a) or (b). From weather satellite images represented as *CoverageSeries*, we can think about extracting trajectories of storms or hurricanes.

The algebra was tested and validated using the TerraLib geospatial software library. An alternative would be to extend an existing spatial database system, such as PostGIS or Oracle Spatial, with the proposed types and functions. However, we have chosen to implement it in a general-purpose library that can access spatiotemporal data from different sources, including databases, files and web services (Figure 12). This approach requires transforming the spatial and temporal information stored in different data file formats and databases into the algebra types. This thesis proposes a strategy to do this transformation based on metadata files. This strategy has been validated using trajectories of moving objects, as presented in Annex B. As future work, we intend to extend and use this strategy for *time series*, *coverage*, *converge series* and *events*, which can also come from different data sources.

6.1. Next Steps Related to Software Implementation

The three next steps related to software implementation are:

- (1) Study new and intuitive ways to display spatiotemporal information and the results of operations over it. As an example, the system shown in Figure 14 displays the sea elephant trajectories through an animation over time. However, there is no interaction between the *time series* generated by the operation *distance* and the *trajectories*. As a starting point, we intend to construct such interactions.
- (2) Develop interfaces with other software tools for statistical analyses and data mining of the proposed data types. As a beginning, we plan

to develop interfaces with the R spacetime package (PEBESMA, 2011) and with the module Weka-STPM (BOGORNY et al., 2011). These interfaces will allow us to use R packages for statistical analysis and to create *semantic* trajectories for data mining using Weka system.

- (3) Provide a mechanism that allows a user to write *scripts* with a set of operations over the proposed data types, as the code examples shown in Section 5.2. Scripts can be written and executed "on-the-fly", without explicit compile and link steps. The idea is to use well-known script languages, such as LUA and Python.

6.2. Comparison with Previous Work

This section presents a comparison between the data types proposed in this thesis and the closest ones defined in previous models. We first compare the Trajectory type with the previous models ISO (2008) and Güting et al. (2000). Trajectory allows geometry deformations over time, whereas the ISO *moving feature model* does not (ISO, 2008). Therefore, the proposed model can cope with applications where entities change their shape, like oil spills and boundary changes in cities. The *moving point* and *moving region* types defined by Güting et al. (2000) always consider a predefined interpolation function, without allowing a user to choose other interpolation methods. Since Trajectory is built from an observation set and an interpolator, we can explicitly choose the most suitable interpolation function for each instance.

The Coverage type is consistent with existing field or coverage definitions (GOODCHILD, 1992; COVA; GOODCHILD, 2002; OGC, 2006b; 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 triangulated irregular network (TIN). Although OGC

coverage definition includes spatiotemporal domains, only coverages with spatial domains are described in its UML class diagrams. OGC coverage with spatiotemporal domain can be mapped to the CoverageSeries type. Besides that, we use an algebra to define our types instead of using UML diagrams. Formal definitions are better than CASE tool diagrams for GIS type specifications (Frank and Kuhn 1995). Such diagrams are not suitable for large problems, where the amount of documentation becomes overwhelming.

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 and Cole (2007), can also be expressed using the Event type. This work focuses on defining an algebra that covers the whole process to obtain events from raw observations. It does not define types of relationships between objects and events neither between events and events. These kinds of relationships, like the ones defined by Worboys and Hornsby (2004) and Galton and Worboys (2005), can be built on top of the proposed model.

REFERENCES

- ALLEN, J. F. Time and time again: the many ways to represent time. **International Journal of Intelligent Systems**, v. 6, n. 4, p. 341-355, July, 1991.
- BOGORNY, V.; KUIJPERS, B.; ALVARES, L. O. ST-DMQL: A semantic trajectory data mining query language. **International Journal of Geographical Information Science**, v. 23, n.10, p. 1245-1276, 2009.
- BOGORNY, V.; AVANCINI, H.; CESAR DE PAULA, B.; KUPLICH, C. R.; ALVARES, L. O. Weka-STPM: a software architecture and prototype for semantic trajectory data mining and visualization. **Transactions in GIS**, v. 15, n. 2, p. 227–248, April, 2011.
- CÂMARA, G.; VINHAS, L.; FERREIRA, K. R.; QUEIROZ, G. R.; SOUZA, R. C. M.; MONTEIRO, A. M. V.; CARVALHO, M. T.; CASANOVA, M. A.; FREITAS, U. M. Terralib: An open-source GIS library for large-scale environmental and socio-economic applications. In: HALL, B.; LEAHY, M. (eds.) **Open source approaches to spatial data handling**. Berlin, Springer-Verlag (ISBN 978-3-540-74830-4), p. 247-270, 2008.
- COUCLELIS, H. People manipulate objects (but cultivate fields): beyond the raster-vector debate in GIS. In: FRANK, A. U.; CAMPARI, I.; FORMENTINI U. (eds). **Theory and methods of spatio-temporal reasoning in geographic space**. Berlin: Springer-Verlag, p. 65-77, 1992.
- COVA, T. J.; GOODCHILD, M. F. Extending geographical representation to include fields of spatial objects. **International Journal of Geographical Information Science**, v. 16, n. 6, p. 509-532, 2002.

EGENHOFER, M.; FRANZOSA, R. Point-set topological spatial relations. **International Journal of Geographical Information Systems**, v. 5, p. 161-174, 1991.

FRANK, A.; KUHN, W. Specifying Open GIS with functional languages. In: INTERNATIONAL SYMPOSIUM ON ADVANCES IN SPATIAL DATABASES, 4., 1999, Portland. **Proceedings...** Springer-Verlag, 1999. v. 951, p.184-195. Egenhofer, M. and Herring, J. (Eds.), Lecture Notes in Computer Science,

FRANK, A. One step up the abstraction ladder: combining algebras - from functional pieces to a whole. In: THE INTERNATIONAL CONFERENCE ON SPATIAL INFORMATION THEORY (COSIT), 1999, Stade, Germany. **Proceedings...** Stade: Springer Verlag, 1999. p. 95-108. Freksa, C. and Mark, D. (Eds.) Lecture Notes in Computer Science.

GALTON, A. Fields and objects in space, time and space-time. **Spatial Cognition and Computation Journal**, v. 4, n. 1, p. 39-68, March 2004.

GALTON, A.; WORBOYS, M. Processes and events in dynamic geo-networks. In: THE INTERNATIONAL CONFERENCE ON GEOSPATIAL SEMANTICS (GEOS 2005), 2005, Mexico. **Proceedings...** Springer Verlag, 2005. V. 3799, p. 45-59. Rodriguez, M. A.; Cruz, I. F.; Levashkin, S.; Egenhofer, M. J. (Eds.). Lecture Notes in Computer Science, Springer-Verlag.

GALTON, A. Experience and history: processes and their relation to events. **Journal of Logic and Computation**, v. 18, n. 3, p. 323-340, June 2008.

GALTON, A.; MIZOGUCHI, R. The water falls but the waterfall does not fall: New perspectives on objects, processes and events. **Applied Ontology**, v. 4, n. 2, p. 71-107, 2009.

GOODCHILD, M. Geographical data modeling. **Computers and Geosciences**, v. 18, n. 4, p. 401-408, 1992.

GRENON, P. ; SMITH, B. SNAP and SPAN: Towards dynamic spatial ontology. **Spatial Cognition and Computation Journal**, v. 4, n. 1, p. 69-104, March 2004.

GÜTING, R. H.; BÖHLEN, M. H.; ERWIG, M.; JENSEN, C. S.; LORENTZOS, N. A.; SCHNEIDER, M.; VAZIRGIANNIS, M. A foundation for representing and querying moving objects. **ACM Transactions of Database Systems**, v. 25, n. 1, 2000.

GÜTING, R. H.; SCHNEIDER, M. **Moving objects databases**. San Francisco, CA: Morgan Kaufmann, 2005. 389 p.

GUTTAG, J. V.; HOROWITZ, E.; MUSSER, D. R. Abstract data types and software validation. **Communications of ACM**, v. 21, 1978.

HORNSBY, K.; EGENHOFER, M. Identity-based change: A foundation for spatiotemporal knowledge representation. **International Journal of Geographical Information Science**, v. 14, n. 3, p. 207-224, 2000.

HORNSBY, K.; EGENHOFER, M. J. Modeling moving objects over multiple granularities. **Annals of Mathematics and Artificial Intelligence**. v. 36, n. 1-2, p.177-194, September, 2002.

HORNSBY, K.; COLE, S. Modeling moving geospatial objects from an event-based perspective. **Transactions in GIS**, v. 11, n. 4, p. 555-573, 2007.

INTERNATIONAL STANDARD ORGANIZATION (ISO). **Geographic information** — temporal schema (ISO 19108). Geneva, 2002.

INTERNATIONAL STANDARD ORGANIZATION (ISO). **Geographic information** — Schema for moving features (ISO 19141). Geneva, 2008.

KNOTTERS, M.; HEUVELINK, G. B. M.; HOOGLAND, T.; WALVOORT, D. J. J. **A disposition of interpolation techniques**. Wageningen, Statutory Research Tasks Unit for Nature and the Environment. 2010.

KUHN, W. Geospatial semantics: why, of what, and how? **Journal of Data Semantics**, v. 3, p. 1-24, 2005.

KUHN, W. A functional ontology of observation and measurement. In: **International Conference on GeoSpatial Semantics (GeoS 2009)**, 2009, Mexico City, Mexico. **Proceedings...** Springer Verlag,, 2009. Lecture Notes in Computer Science.

LAUBE, P.; IMFELD, S.; WEIBEL, R. Discovering relative motion patterns in groups of moving point objects. **International Journal of Geographical Information Science**, v. 19, n. 6, p. 639–668, July, 2005.

LIU, Y.; GOODCHILD, M. F.; GUO, Q.; TIAN, Y.; WU, L. Towards a general field model and its order in GIS. **International Journal of Geographical Information Science**, v. 22, n. 6, p. 623-643, 2008.

MARK, D.; EGENHOFER, M. J.; BIAN, L.; HORNSBY, K.; ROGERSON, P.; VENA, J. Spatio-temporal GIS analysis for environmental health using geospatial lifelines. In: INTERNATIONAL WORKSHOP ON GEOGRAPHY AND MEDICINE (GEOMED'99), 2., 1999, Paris. **Proceeding...** Paris, France, 1999.

MENNIS, J. Multidimensional map algebra: design and implementation of a spatiotemporal GIS processing language. **Transactions in GIS**, v. 14, n. 1, p. 1-21, 2010.

OPEN GEOSPATIAL CONSORTIUM (OGC). **OpenGIS implementation specification for geographic information - simple feature access - part 1: common architecture**. Open Geospatial Consortium, 2006a. Available at: <www.opengeospatial.org/>. Access at: 09/11/2012.

OPEN GEOSPATIAL CONSORTIUM (OGC). OpenGIS abstract specification topic 6: **Schema for coverage geometry and functions**. Open Geospatial Consortium, 2006b. Available at: <www.opengeospatial.org/>. Access at: 09/11/2012.

PEBESMA, E. **Classes and methods for spatio-temporal data in R: the spacetime package**. Munster, Germany: Institute for Geoinformatics, University of Munster,. 2011. Available at: <<http://cran.R-project.org/web/packages/spacetime/vignettes/spacetime.Pdf>>. Access at: 09/11/2012.

PELEKIS, N.; THEODOULIDIS, B.; KOPANAKIS, I.; THEODORIDIS, Y. Literature Review of Spatio-Temporal Database Models. **The Knowledge Engineering Review**, v. 19, n. 3, p. 235-274, 2004.

PELEKIS, N.; FRENTZOS, E.; GIATRAKOS, N.; THEODORIDIS, Y. HERMES: Aggregative LBS via a trajectory DB engine. In: ACM SIGMOD' 08 CONFERENCE, 2008, Vancouver, BC, Canada. **Proceedings...** Vancouver: ACM, 2008.

PEUQUET, D. J.; DUAN, N. An event-based spatiotemporal data model (ESTDM) for temporal analysis of geographical data. **International Journal of Geographical Information Science**, v. 9, n. 1, p. 7-24, 1995.

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.; MONTEIRO, A. M. An entomological surveillance system based on open spatial information for participative dengue control. **Anais da Academia Brasileira de Ciências**, v. 81, p. 655-662, 2009.

RIGAUX, P.; SCHOLL, M.; VOISARD, A. **spatial databases: with application to GIS**. San Francisco, USA: Morgan Kaufmann, 2002.

SAKR, M. A.; GÜTING, R. H. Spatiotemporal pattern queries. **Geoinformatica**, v. 15, p. 497–540, 2011.

SINTON, D. 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, v. 7, p. 1-17, 1978.

SPACCAPIETRA, S.; PARENT, C.; DAMIANI, M.; MACEDO, J. A. F.; PORTO, F.; VANGENOT, C. A conceptual view on trajectories. **Data & Knowledge Engineering**, v. 65, p. 126-146, 2008.

WORBOYS, M. F. A Unified Model for Spatial and Temporal Information. **The Computer Journal**, v. 37, n. 1, 1994.

WORBOYS, M.; HORNSBY, K. From Objects to Events: GEM, the Geospatial Event Model. In: INTERNATIONAL CONFERENCE ON GISCIENCE, 3., 2004, Berlin. **Proceedings...**Spring Verlag, 2004. V. 3234, p. 327-343. Egenhofer, M.; Freska, C. and Miller, H. (eds.), Lecture Notes in Computer Science.

WORBOYS, M. F.; DUCKHAM, M. **GIS - a computing perspective**. 2. ed. Boca Raton: CRC Press, 2004.

WORBOYS, M. Event-oriented approaches to geographic phenomena. **International Journal of Geographical Information Science**, v. 19, n. 1, p. 1-28, 2005.

YUAN, M. Use of a Three-domain representation to enhance gis support for complex spatio-temporal queries. **Transactions in GIS**, v.3 n.2, p.137-159, 1999.

ANNEX A – TOWARDS A DYNAMIC GEOSPATIAL DATABASE MODEL

This annex presents a paper published in the *International Conference on Emerging Databases - EDB 2011*, Incheon, Korea (Ferreira et al, 2011):

Towards a Dynamic Geospatial Database Model

Karine Reis Ferreira,
Gilberto Camara,
Antônio Miguel Vieira Monteiro

DPI – Image Processing Division, INPE – National Institute for Space Research,
Av. dos Astronautas 1758, 12227-001 São José dos Campos, SP, Brazil
{karine, gilberto, miguel}@dpi.inpe.br

Since most existing spatio-temporal database models are specific to meet a particular set of applications, there is a need for a more general one which is not application-oriented and can be used for a new generation of dynamic geographical information systems. Thus, this work aims to identify a set of requirements for a new database model, called Dynamic Geospatial Database Model (DyGeo Model), able to represent and query different geospatial data dynamics and so to support different kinds of spatio-temporal applications. These requirements were defined based on an analysis of distinct geospatial data dynamics and on a critical review of ten spatio-temporal database models proposed in literature during the past two decades.

Key Words: dynamic geospatial data, spatio-temporal database model, dynamic geographic information systems.

1. INTRODUCTION

The recent technological advances in geospatial data collection, such as Earth observation and GPS satellites, wireless and mobile computing, radio-frequency identification (RFIDs), and sensor networks, have motivated new types of applications which handle spatio-temporal information. Examples include animal tracking and oil spill on the ocean, land parcel changes, as well as environmental change monitoring based on satellite images. To meet this demand, it is necessary to represent dynamic geospatial information in spatial databases and geographical information systems (GIS).

Static geospatial information is represented in GIS following well-established ideas. These ideas include object-based and field-based models [1], vector and raster data structures, topological operators [2], spatial indexing as well as spatial joins and operations [3]. In recent years, database management systems (DBMS) have been extended to handle 2D static geospatial information and there has been a major effort to standardize basic components for such data [4].

However, there is no consensus on how to represent spatio-temporal information in computational systems. According to Worboys [5], there are four stages in introducing temporal capacity into GIS and most current proprietary technologies are in stage zero, that is, they do not deal with spatio-temporal information. In GIS literature, there are many proposals of spatio-temporal database models. Nevertheless, Pelekis et al. [6] consider that most existing models are application-oriented, focusing on particular aspects of spatio-temporal data. They are not general enough to be a basis for a new generation of dynamic geographical information systems.

Therefore, this work aims to identify a set of requirements for a more general and not application-oriented model called Dynamic Geospatial Database Model (DyGeo Model). The main idea is to define a new model able to represent and query different geospatial data dynamics and then to support different kinds of spatio-temporal applications. Following this idea, the first phase aims to identify a consistent set of requirements that the DyGeo model must meet.

The DyGeo model requirements have been identified based on an analysis of distinct geospatial data dynamics and on a critical review of ten spatio-temporal database models. So, section 2 analyses distinct geospatial data dynamics and illustrates each one with real spatio-temporal applications. Section 3 provides a critical review of ten spatio-temporal database models proposed in GIS literature during the past two decades. They are well-known models which have high number of citations in the literature. The DyGeo model requirements are presented in section 4 and section 5 concludes this work.

2. DYNAMIC GEOSPATIAL DATA

Based on the dichotomy, *geo-objects* and *geo-fields*, to represent geospatial data [1], dynamic geospatial data can be represented by either (1) Geo-objects which vary over time or (2) Geo-fields which change over time. In the first representation, there are three cases: (1.1) Geo-object whose geometry is fixed but its non-spatial attributes change over time; (1.2) Geo-object whose geometry changes discretely over time and whose non-spatial attributes also can change; and (1.3) Geo-objects whose geometry changes continuously over time and whose non-spatial attributes also can change.

Since the spatial component of a geo-object is represented by geometries, such as polygons, lines and points, this work uses the term “geometry” for the geo-object spatial component. Besides that, the term “non-spatial attributes” refers to features associated to geo-object which are represented by primitive data types, such as numbers and texts.

Regarding geo-objects which change over time, the difference between discrete and continuous geometry changes is pointed out by Galton [7] when he explains the difference between bona fide and fiat object behavior over time. Bona fide objects are grounded in features of physical reality, such as rivers and forest regions, and fiat objects are the artificial products of human cognitive acts, such as municipality limits and land parcels. So, he says “Both these objects might change over time, but typically the bona fide entity will undergo gradual change whereas the fiat entity undergoes sudden change (as a result of the boundary being redrawn from time to time).” In this work, “gradual change” is called continuous change and “sudden change” is called discrete change.

Guting and Schneider [8] also talked about this difference, saying that “Regarding kinds of changes, a major distinction concerns discrete changes and continuous changes. Classical research on spatio-temporal database has focused on discrete changes of all the spatial entities. In contrast, the term moving objects emphasizes the fact that geometries change continuously.”

In order to illustrate the main features of each geospatial data dynamic presented above, the following sections present four real applications and their demands on representing and querying dynamic geospatial information. They are: (1) Dengue Fever Monitoring; (2) Municipal Management; (3) Movement Monitoring; and (4) Amazon Deforestation Monitoring.

Universities and research institutes in Brazil have been involved in a cooperative project called SAUDAVEL which aims at building a surveillance system to control, warn and intervene in epidemic and endemic diseases, like **Dengue Fever** and **Leptospirosis** [9]. The central experiment of this project is being carried out in Recife, Brazil. Mainly, it consists in giving out egg traps for *Aedes aegypti* and *Aedes albopictus* mosquitoes in different locations around the city and in counting the number of mosquito eggs found in each trap weekly. Then, this data is

processed together with environmental information, resulting in risk maps for public health interventions.

In this first application, each egg trap can be considered as a fixed geosensor, that is, a sensor which collects information at different times associated to a fixed location. The location of each trap does not change over time, only its attributes, such as number of mosquito eggs. So, each egg trap can be represented by a geo-object whose geometry is fixed but its non-spatial attributes change over time. Besides that, some important queries associated to this application are: (1) *What was the monthly mosquito egg average for each trap?* (2) *Which month presented the biggest number of mosquito eggs?* (3) *When and where were more than 80 mosquito eggs collected by each trap?* (4) *How many eggs were collected in the summer season?* (5) *Which district had the biggest/smallest number of mosquito eggs?*

Figure 1 shows a set of egg traps (represented by red points) in a Recife's district called "Engenho do Meio" and a time series generated by the egg trap EM124. This time series represents the number of collected eggs (axis y) by date (axis x).

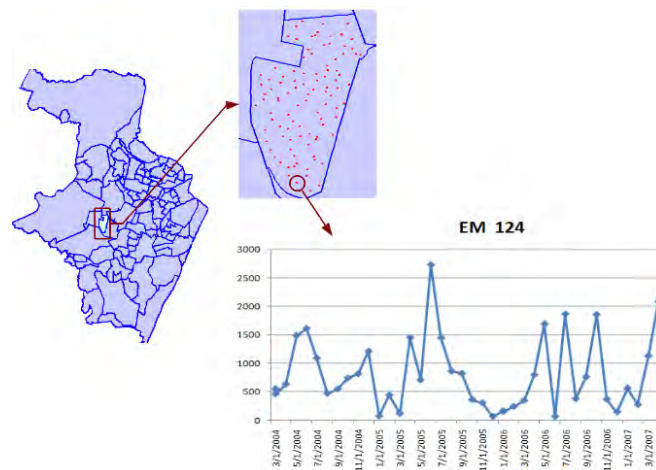


Figure 1. SAUDAVEL egg traps.

Municipal management applications deal with municipality related issues, such as urban land parcels and municipal limit changes. In this application, each urban parcel boundary as well as each municipal limit change discretely over time and their non-spatial attributes can also vary (e. g. the municipal government and the parcel owner). As an instance, Figure 2 shows changes in Rondônia's municipality limits. In this example, three municipalities "Costa Marques", "São Francisco do Guaporé" and "Seringueiras" had the same limits from 2001 to 2004, and then, on the first day of 2005 they suddenly changed due to new laws.

Movement Monitoring refers to applications which monitor and analyze object motions, such as animal, vehicle and person movement. These kinds of applications consist in tracking objects by getting their locations as well as other

information such as animal temperature and vehicle velocity at different times. In this case, the object locations vary continuously over time and the concept of trajectory is very important. Some related queries are: (1) *Where was object o_1 at time t_5 ?* (2) *When did object o_1 enter a specific region r_{10} and how long did it stay in this region?* (3) *When and where did objects o_1 and o_2 meet each other (considering a meeting when the distance between two objects is less than 2 meters)?* (4) *Where and when was there a spatio-temporal cluster of objects?*



Figure 2. Rondônia's municipality limits in 2001 (left picture) and in 2005 (right picture). Legend: blue polygon is "Costa Marques" municipality; yellow is "São Francisco do Guaporé" and green is "Seringueiras".

The Brazilian **Amazon deforestation** has been monitored since 1988 by National Institute for Space Research (INPE) through a project called PRODES. It is responsible for calculating Amazon deforestation and for identifying deforested regions in each year through satellite images, by using a well-established methodology [10]. Each deforested region evolves continuously and nonlinearly over time and this evolution must be represented in order to allow a specialist to refine its analysis by recognizing patterns of deforested regions [11] and how these patterns evolve over time [12]. A real example of a deforested region evolution is shown in Figure 3.

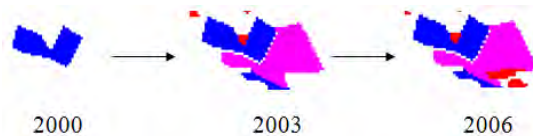


Figure 3. Evolution of a deforested area. Source: [10].

In this case, each deforested region can be represented by a geo-object whose geometry changes continuously over time and some important queries are: (1) *What was the state of a specific deforested region like in 2003?* (considering that this specific deforested region was observed in 2002 and in 2005, but not in 2003) (2) *What was the area and perimeter variation over time of a specific deforested region?* (3) *How did a specific deforested region evolve over time between 2000 and 2008?* (4) *How did the deforested regions that started less than 2 kilometers*

far from river r1 evolve over time? (5) When did a specific deforested region reach municipality x?

Besides the polygonal representation of each deforested region, PRODES project also generates sets of classified images to represent deforestation process. Figure 4 shows an example of the deforestation process in a specific region in Amazon, based on four classified images from different times. These images can be better represented by a geo-field which change/evolve over time since geo-object concept does not exist in this case.

Some important queries associated to it are: (1) *Given a pixel or cell, how has the forest status been varying in this cell over time?* (2) *What was the deforestation in this specific region like in 2001?* (considering that there is no classified image from 2001.) (3) *How many hectares were deforested in this specific region over time?*

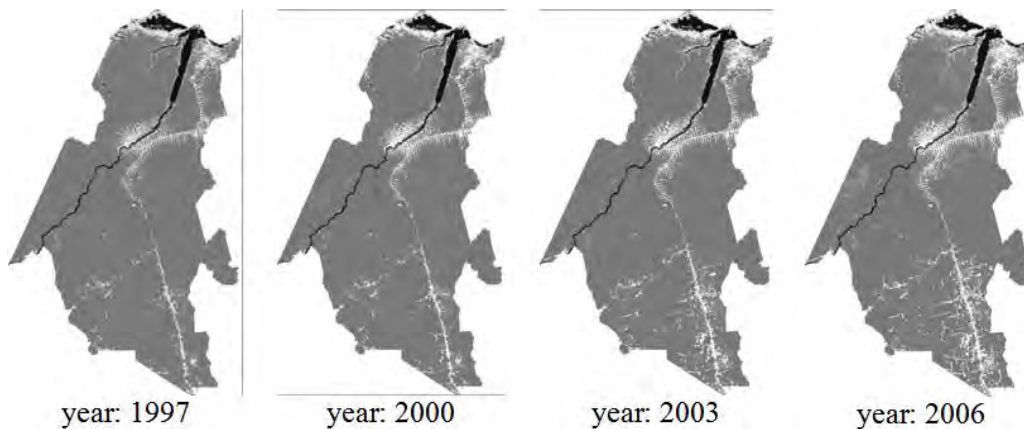


Figure 4. Sequence of four classified images from different years that represent the deforestation process in a specific region in Amazon rainforest. In these images, there are basically three classes: river (dark gray), forest (gray), and deforested area (light gray). Source: [10].

3. SPATIO-TEMPORAL DATABASE MODELS: A CRITICAL REVIEW

During the past two decades, many spatio-temporal database models have been proposed in GIS literature. This section presents a critical review of ten models which propose an ontology of space and time and its representation through data types, relationships and operations thereon. They are well-known models which have high number of citations in GIS literature and are shown in Figure 5.

The Time-Slice Snapshot [13] is the simplest model of them. This model works with a set of snapshots, where each one is a raster layer which represents a state of the real world at a given time, like a photograph. Each snapshot is a collection of

temporally homogeneous units and there are no explicit temporal relations among snapshot. It has two main limitations: (1) operations among snapshots must compare them exhaustively; and (2) redundant storage because a complete snapshot is produced at each time slice, duplicating all unchanged data.

The Space-Time Composite (STC) model [13] is an evolution of Snapshot model, by considering vector objects which change over time instead of raster time-slice layers. The mechanics of this model begin with a base layer which represents the objects at some starting time. After this, each change decomposes the space over time into increasingly smaller fragments (objects with geometries) with its own distinct history. Despite being very simple, it is important because it introduces the idea of representing spatial objects which vary over time.

The Unified Spatio-Temporal Object Model (STOM), proposed by Worboys [14], defines basically two spatio-temporal data types, *ST-simplexes* and *ST-complexes*, and a set of operations over them, such as *ST-Union*, *ST-Intersection* and *ST-Difference*. *ST-simplex* is an ordered pair $\langle S, T \rangle$, where *S* is a simplex data type and *T* is a bitemporal element (BTE). A simplex is either a single point, or a finite straight line segment or a triangular area. And BTE is a temporal data type composed of event and transaction time. At last, a *ST-complex* is a finite set of *ST-simplexes*. The main disadvantage of the STOM model is not to consider changes in object attributes, that is, in the textual and numerical extents of geographical objects.

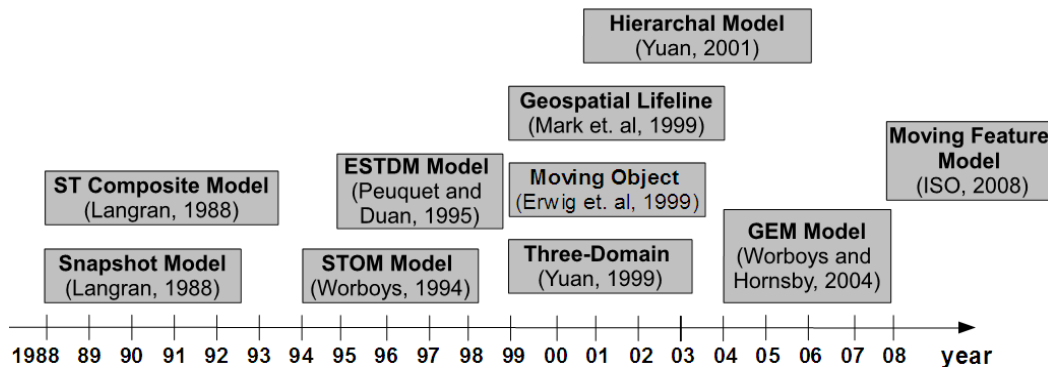


Figure 5. Spatio-temporal database models.

The main idea of Event oriented Spatio-Temporal Data Model (ESTDM) [15] is to group changes by time of occurrence, ordering changes in locations within a predetermined geographical area. The time associated with each change, called event, is stored in increasing order from initial time t_0 to the latest time t_n . The set of changes C_i recorded for any time t_i consists of the set of each location (x, y) which changed since t_{i-1} , and its new value v . Its two main characteristics are: (1) the events are recorded when changes occur, that is, in any temporal resolution; (2) a value v is recorded only when it is different from the last one found along the scan line. So, this model does not have the

two limitations of Snapshot model because it stores only the changed cells by each event. Besides that, it defines a very simple event concept, without exploring concepts related to it, such as, semantics or relationships.

The Three-domain model [16] mainly focuses on how to represent geo-objects which vary over time in a relational database system by using normalized tables and a spatial graph as well as on how to query them by using SQL language. The proposed database schema consists in four tables, one for each domain (semantic, temporal and spatial) and another for the domain link. It can also be implemented in spatial DBMS, as PostGIS and Oracle Spatial, by using its support to deal with spatial information. It is a simple model, which does not define spatio-temporal data types and operations. It only uses the data types and query language provided by DBMS.

Moving Object defines a robust algebra, data types and operations, in two levels of abstraction, abstract and discrete, to deal with moving objects. Moving Object refers to entities whose geometries change continuously over time, such as, cars, aircraft, ships, mobile phone users, polar bears, hurricanes, forest fires, or oil spills in the sea [8]. The authors propose an algebra with two main data types, *moving points* and *moving regions*, and a set of auxiliary types, such as *moving real* and *moving int*. Besides that, this algebra defines a set of operators over these data types, such as *trajectory*, *distance*, *direction*, and *velocity*. Its principal disadvantage is not to consider geo-fields which vary over time. For instance, a hurricane must be represented in this model as a moving region. Nevertheless, in some applications, the best representation of a hurricane is a geo-field which varies over time and not an object. As a prototype of spatio-temporal database, the moving object model was implemented in SECONDO, a database system that is extensible by algebra modules (<http://dna.fernuni-hagen.de/Secondo.html/>).

The Geospatial Lifeline Model [17] defines a geospatial lifeline concept which models an individual's movement as a time-stamped record of locations. The basic element of lifeline data is a triple $\langle Id, Location, Time \rangle$, where *Id* is a unique identifier of the individual, *Location* is a spatial descriptor (such as a coordinate pair, a polygon and a street address), and *Time* is the time stamp when the individual was at that particular location. Besides that, this model proposes different types of trajectories or movement approximations, such as, *threads*, *beads*, *necklaces*, and *convex hulls* [18]. Depending on the desired granularity and on the application type, distinct types of trajectories are essential. For example, in animal tracking, the convexhull trajectory is necessary in order to define a habitat. So, although this model does not define operations over moving objects, it defines important different types of trajectories. In the Moving Object model only the linear or thread trajectory is extracted from moving points (through the operator trajectory).

The Hierarchal model, proposed by Yuan [19], provides an interesting way of organizing, using hierarchical layers, dynamic geographical phenomena which possess both field and object characteristics. It is based on a sequence of snapshots called state layers. Therefore, it has redundant storage problem like the Snapshot model. Besides the snapshots, this model also stores the objects which represent the phenomena. These objects are extracted from the state layers. Thus, these two representations of

phenomena, geo-fields and geo-objects, are used to improve the spatio-temporal query processing and operations. Finally, this model also defines the concepts of event and process only to organize the data layers in different levels.

The Geospatial Event Model (GEM), proposed by Worboys and Hornsby [20], is interesting because it introduces an event concept and relationships between events and geo-objects in a model based on spatial objects. It defines two kinds of relationships, *object-event* and *event-event*, following the idea that an event can affect or be associated to one or more objects or events of different types. Some examples of object-event relationships are *splitting* and *merger* (An event that creates/destroys a boundary between objects). Some examples of event-event relationship are *initiation* and *termination* (The occurrence of event A starts / terminates event B). However, it is a model which defines only data types but not operations over them.

The Moving Feature Model, proposed by the International Organization for Standardization (ISO), defines a conceptual schema for moving feature [21]. The term *feature* refers to an abstraction of real world phenomena and *moving feature* refers to features whose geometries move over time. This schema includes a set of classes, attributes, associations, and operations which provides a common conceptual framework to deal with feature geometry which moves as a rigid body. Therefore, it supports changes of location, translation and rotation of a feature, but not other change types, such as, the feature deformation and changes in non-spatial attributes of a feature. The main advantage of this model is to define a generic type called one-parameter geometry which represents the variation of feature geometry with respect to any single variable, such as pressure, temperature, or time. However, its main disadvantages are not to consider feature geometry deformation and changes in feature non-spatial attributes.

4. TOWARDS A DYNAMIC GEOSPATIAL DATABASE MODEL

Since most existing spatio-temporal database models are specific to meet a particular set of applications, there is a need for a more general one which is not application-oriented and can be used for a new generation of dynamic geographical information systems. This new model must be able to represent and query distinct geospatial data dynamics and so to support different kinds of spatio-temporal applications. Therefore, based on the different geospatial data dynamics shown in section 2 and on the reviewed models in section 3, this section presents a set of requirements which this new model, called Dynamic Geospatial Database Model (DyGeo Model), must meet.

Sections 4.1 to 4.5 describe requirements that are related to “*what*” the DyGeo model must represent and query. Sections 4.6 and 4.7 describe requirements related to “*how*” it can be clearly and usefully defined and formalized.

4.1 Represent both Geo-Fields and Geo-Objects which Vary over Time

The models presented in section 3 can be grouped in two classes: (1) models which are specific to represent geo-fields that change over time and (2) models which are specific to represent geo-objects that change over time. Still in the second class, there is a subset of models specialized in representing geo-objects whose geometries change continuously over time. This classification is shown in Table 1.

Considering these models and the applications presented in section 2, the question that arises is: What database model is able to support the four applications, that is, able to represent and query the information generated by them? Unfortunately, there is no single model general enough to support them. So, we could use different models, one for each application. For example, the Moving Object model to support the Movement Monitoring application, STOM model to Municipal Management and ESTDM Model to represent the images of Deforestation Monitoring. However, in this case, how to combine dynamic geospatial data from different applications in our analysis? How to mix, for instance, trajectories of animals in the Amazon forest with the deforestation process (represented by a set of classified images at different times)? There is no model that defines this kind of operations.

Therefore, there is a lack of a single model able to represent and query geo-fields as well as geo-objects which vary over time, considering discrete and continuous geometry and non-spatial attribute changes of geo-objects. Representing these different geospatial data dynamics, the DyGeo model should be able to support different kinds of applications, such as the four applications presented in section 2.

Table 1. Classification of existing spatio-temporal models.

Geo-Fields which change over time	Geo-Objects which change over time	
	Discrete geometry change	Continuous geometry change
Snapshot Model, ESTDM Model, and Hierarchal Model	STC Model, STOM Model Three-domain Model, and GEM Model	Moving Object Model, Geospatial lifeline, and Moving Feature Model

4.2 Define Operations between Geo-Fields and Geo-Objects which Vary over Time

Besides representing and querying geo-fields as well as geo-objects which vary over time, the DyGeo model must define operations between them. For example, considering the monitoring of animals in the Amazon forest (represented by geo-objects whose geometries change continuously over time) and the deforestation process (represented by geo-fields which change over time), the model should be able to answer questions

like: (1) *When did animal a_1 go into forested areas?* (2) *How long did animal a_1 stay in deforested areas? And what was its mean temperature during this period?*

Another example of question which requires operations between geo-fields and geo-objects which vary over time is “*How many hectares were deforested in each municipality?*”, presented in section 2.1. In this case, the answer must take into account geo-objects whose geometries vary discretely over time (municipality limit changes) as well as geo-fields which change over time (deforestation process).

In order to answer queries between geo-fields and geo-objects which vary over time, the DyGeo model must define a set of operators which receive both as parameters and combine them. For example, to answer the question “*When did animal a_1 go into forested areas?*”, the model must provide an operator able to compute the class, deforested or forest (see Figure 4), for each animal location. So, if the animal a_1 were in a deforested area at time t_5 and in a forest area at time t_6 , where $t_5 < t_6$, we could conclude that it went into a forest area between t_5 and t_6 .

4.3 Define Spatio-Temporal Interpolators

Since computational systems are discrete, continuous processes are often represented in them as a set of discrete measures. For example, the tracking of an animal is represented by a set of its locations, each one measured in a specific time. Likewise, the Amazon deforestation process is measured by detecting deforested regions at distinct times.

Spatio-temporal interpolators for geo-objects whose geometries vary continuously over time are essential in order to estimate a space in a specific time when there is no measurement available about it. For instance, in the second application, shown in Figure 2, each municipal limit changes discretely over time. So, in order to answer the question “*what was the limit of Seringueiras municipally in 2003?*” we do not need to interpolate its limits between 2001 and 2005 because it had the same limit from 2001 to 2004. Otherwise, in the fourth application presented in Figure 3, in order to answer the question “*what was the deforested region like in 2002?*” we do need to interpolate it between 2000 and 2003 because a deforested region evolves continuously over time.

In addition to that, spatio-temporal interpolators for geo-fields which change over time are also necessary. For example, considering the images presented in Figure 4, in order to answer the question “*what was the deforestation in this specific region like in 2001?*” we need to interpolate the images between 2000 and 2003, because we assume that the deforestation process is continuous over time. In this case, the interpolators can consider pixel or cell neighborhoods and how they evolve over time.

Actually, spatio-temporal interpolators are mechanisms to approximate discrete measures to continuous processes. There are different ways to do it. For example, Hornsby and Egenhofer [18] propose distinct trajectory approximations for moving point, such as linear, necklace and convexhull, as shown in Figure 6.

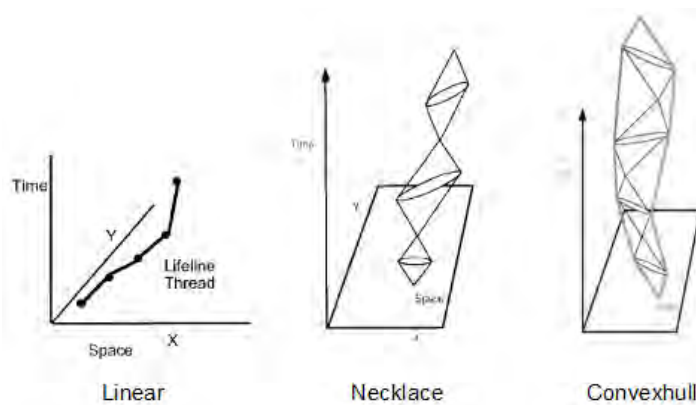


Figure 6. Types of trajectory approximations. Source: [18].

Therefore, the DyGeo model must provide distinct kinds of spatio-temporal interpolators for geo-objects whose geometries change continuously over time as well as for geo-fields which vary over time. These interpolators must be internally used by most of DyGeo operations and must be selected based on the characteristics of the data, such as the space and time granularities.

4.4 Represent Semantics of Changes

In some kinds of applications, besides representing spatial changes over time, it is necessary to associate semantics to them. For example, in the Municipal Management application, a new urban land parcel can be created through the merging of two old ones or through the splitting of one parcel. In this case, the application can be interested in knowing *what caused* a land parcel merging or splitting. In the animal tracking application, an animal can die and its sensor can be used to monitor another animal. Or an animal can be hurt and this state can interfere in its trajectory. So, in this application, a user can be interested in knowing *when* and *why* an animal died as well as *when* and *why* it was hurt.

Regarding the Amazon deforestation monitoring, there are works which study different patterns of deforested regions and how they evolve over time [11] [12]. They try to understand *who* and *what causes* some specific patterns of deforested areas. For instance, new small settlements emerge; large farms increase their agricultural area at the expense of the forest and, then, farmers buy land from small settlers to increase their property for large-scale agriculture and extensive cattle ranching.

It is important to note that each application has its own semantic scope. Therefore, the DyGeo model must provide mechanisms that allow each application to define and represent its change semantics and to consider them in its queries.

4.5 Represent Geospatial Processes and Relationships among them

Nowadays, one research challenge in geospatial science is to define and represent dynamic geospatial processes and relationships among them [5]. In GIS literature, there

are many distinct definitions of geospatial process, depending on the application domain.

So, the DyGeo model must provide a mechanism that allows an application user to define geospatial processes based on the application domain and, then, relationships among them. For example, a user can define the Amazon deforestation as a geospatial process and the migration of an animal species as another process. And then, the user can identify relationships between these two processes, such as “*the deforestation process starts up the migration of an animal species in the Amazon*”.

Among the models presented in this work, only the GEM Model defines a set of event-event relationships, such as, *initiation* and *termination* (The occurrence of event A starts / terminates event B).

4.6 Describe the Model by Using Algebraic Specifications

Algebraic specifications provide a mathematical framework for describing abstract data types. The main advantage of this framework is the capacity of formally describing required types and operations thereon, independent of programming language and implementation. Besides that, it is possible to specify semantics of operations [22]. So, the DyGeo model should be clearly and formally defined by using well-known algebraic formalisms.

Among the models presented in section 3, the Moving Object Model is the clearest and best formally defined. It utilizes a many-sorted algebra to express its spatio-temporal data types and operations.

4.7 Define the Model in Two Levels of Abstraction, Conceptual and Physical

The ANSI-SPARC (*American National Standards Institute, Standards Planning And Requirements Committee*) proposes a database architecture based on three levels: **external**, **conceptual** and **internal** [23].

The external level (user view) describes a part of the database that is relevant to a particular user. It looks at the world from a particular perspective, for a particular purpose. The conceptual or logical level is a representation of what data is stored within the whole database and how it is inter-related. It focuses on describing the model concepts without worrying about the way the data is physically stored in computational systems. It is independent of hardware and software. At last, the physical or internal level is a low-level representation of the entire database. It describes how the database is physically represented on the computer system.

The description of a database model in all these levels is very useful and important, mainly the conceptual and physical ones. The conceptual model is important to describe the main concepts in an understandable way. While the physical one presents implementation details and is useful to describe how to materialize a conceptual model in a computational system. Since the external level is related to particular user views

and is based on the conceptual model, we believe that it is not essential to understand the whole database model.

To wrap up this section, Table 2 shows what requirements are met by each model presented in section 3.

Table 2. Requirements versus existing spatio-temporal database models.

	Requirements						
	4.1	4.2	4.3	4.4	4.5	4.6	4.7
Snapshot							
ST Composite							
STOM						✓	
ESTDM					✓		
Three Domain							✓
Moving Object			✓			✓	✓
Geospatial Lifeline			✓				
Hierarchal Model					✓		
GEM				✓	✓		
Moving Feature							✓

5. CONCLUSION

This work presents a set of requirements for a more general and not application-oriented model, called Dynamic GeoSpatial Database Model (DyGeo Model), able to represent and query different geospatial data dynamics and so to support different kinds of spatio-temporal applications. These requirements are identified based on an analysis of distinct geospatial data dynamics and on a critical review of some spatio-temporal database models proposed in literature during the past two decades. This work reviews ten models which define space and time representations through data types, relationships and operations thereon. They are well-known models which have high number of citations in GIS literature.

We believe that the phase of requirements gathering, based on an analysis of distinct geospatial data dynamics as well as on a critical review of the existing spatio-temporal database models, is the first step towards the DyGeo model. It is crucial before actually defining a consistent model. For now on, the next step is to formally define the DyGeo algebra taking the requirements identified in this work as a basis for its data types and operators.

REFERENCES

- [1] Couclelis, H., "People Manipulate Objects (but Cultivate Fields): Beyond the Raster-Vector Debate in GIS," *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, Frank, A., et al. (eds.). Springer-Verlag, Berlin. pp. 65-77. 1992.

- [2] Egenhofer, M., Franzosa, R., "Point-Set Topological Spatial Relations," *International Journal of Geographical Information Systems*, vol. 5, pp. 161-174. 1991.
- [3] Rigaux, P., Scholl, M., Voisard, A., *Spatial Databases with Application to GIS*. Morgan Kaufman, San Francisco, 2002.
- [4] Open Geospatial Consortium (OGC), "OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture" [online]. (OGC 06-103r3). Available from: www.opengeospatial.org [Accessed 27/01/2009]. 2006.
- [5] Worboys, M., "Event-oriented approaches to geographic phenomena," *International Journal of Geographical Information Science*, vol. 19, pp. 1-28. 2005.
- [6] Pelekis, N., Theodoulidis, B., Kopanakis, I., Theodoridis, Y., "Literature review of spatio-temporal database models," *The Knowledge Engineering Review*, vol. 19. 2004.
- [7] Galton, A., "Fields and Objects in Space, Time and Space-Time," *Spatial Cognition and Computation Journal*, vol. 4, pp. 39-68. 2004.
- [8] Güting, R. H., Schneider, M., *Moving Objects Databases*. Morgan Kaufmann, New York, 2005.
- [9] 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., Monteiro, A. M., "An Entomological surveillance system based on open spatial Information for participative Dengue control," In *Anais da Academia Brasileira de Ciências*. vol. 81, pp. 655-662. 2009.
- [10] INPE: "Monitoramento da Floresta Amazônica Brasileira por Satélite (Monitoring the Brazilian Amazon Forest by Satellite)" [online]. *Report, São José dos Campos: INPE*, Available from: <http://www.obt.inpe.br/prodes> [Accessed 05/04/2010]. 2008.
- [11] Silva, M., Camara, G., Souza, R. C., Valeriano, D., Escada, I., "Mining Patterns of Change in Remote Sensing Image Databases," In *15th IEEE International Conference on Data Mining*. Houston, TX, USA, 2005.
- [12] Mota, J., Camara, G., Escada, I., Bittencourt, O., Vinhas, L., "Case-Based Reasoning for Eliciting the Evolution of Geospatial Objects," In *Conference on Spatial Information Theory COSIT 2009*, Aber Wrach, France, 2009.
- [13] Langran, G., Chrisman, N. R., "A Framework For Temporal Geographic Information," *The International Journal for Geographic Information and Geovisualization*, vol. 25. 1988
- [14] Worboys, M. F., "A Unified Model for Spatial and Temporal Information," *The Computer Journal*, vol. 37. 1994.
- [15] Peuquet, D. J., Duan, N., "An event-based spatiotemporal data model (ESTDM) for temporal analysis of geographical data," *International Journal of Geographical Information Science*, vol. 9, pp. 7-24. 1995.
- [16] Yuan, M., "Use of a Three-Domain Representation to Enhance GIS Support for Complex Spatio-temporal Queries," *Transaction in GIS*, vol. 3, pp. 137-159. 1999.
- [17] Mark, D., Egenhofer, M. J., Bian, L., Hornsby, K., Rogerson, P., Vena, J., "Spatio-temporal GIS analysis for environmental health using geospatial lifelines," In *2nd International Workshop on Geography and Medicine (GEOMED'99)*. Proceeding... Paris, France. 1999.
- [18] Hornsby, K., Egenhofer, M. J., "Modeling Moving Objects over Multiple Granularities," In *Annals of Mathematics and Artificial Intelligence*, vol. 36, pp. 177-94. 2002.
- [19] Yuan, M., "Representing Complex Geographic Phenomena in GIS," *Cartography and Geographic Information Science*, vol. 28, pp. 83-96. 2001.
- [20] Worboys, M. F., Hornsby, K., "From objects to events: GEM, the geospatial event model," In *Third International Conference on GIScience*, pp. 327-343, Springer-Verlag, Berlin, 1994.
- [21] International Standard Organization (ISO), "Geographic information - Schema for moving features (ISO 19141)", 2008.

- [22] Guttag, J. V., Horowitz, E., Musser, D. R., “Abstract Data Types and Software Validation,” *Communications of ACM*, vol. 21. 1978.
- [23] Date, C. J., *An Introduction to Database Systems*. Addison-Wesley, Reading , 2004.

ANNEX B – MOVING OBJECTS AND SPATIAL DATA SOURCES

This annex presents a paper accepted in September 2012 for publication in the *Revista Brasileira de Cartografia*:

MOVING OBJECTS AND SPATIAL DATA SOURCES

Karine Reis Ferreira¹
Lúbia Vinhas¹
Antônio Miguel Vieira Monteiro¹
Gilberto Câmara¹

¹Instituto Nacional de Pesquisas Espaciais
Divisão de Processamento de Imagens
Av. do Astronautas, 1758 Jardim da Granja São José dos Campos-SP 12227-010
{karine, lubia, miguel, gilberto}@dpi.inpe.br

ABSTRACT

Moving object is a well-established concept in geographic information system (GIS) science. It is an entity whose spatial position or extent changes continuously over time. Some examples are cars, animals and deforested regions. Nowadays, there is a growing demand for GIS tools that are able to handle and analyze moving objects. Most existing spatial file formats (e.g. KML and GML) and database systems (e.g. PostGIS) represent spatial and temporal information using structures and types predefined in specifications written by the International Organization for Standardization (ISO) and the Open Geospatial Consortium (OGC). However, in these specifications, there is nothing about moving object representation in data files or databases. Each data producer adopts its own format to do it. Therefore, this work proposes an interoperable strategy to translate spatial and temporal information stored in different data sources into moving object trajectories for further analyses. The proposed approach is based on the processing of an additional metadata file that describes *how* moving objects are stored in a particular data source. Grounded on this strategy, we have built a new software module for moving object analysis in a geographical library called TerraLib. This module architecture is also described in this paper.

Keywords: Geographical information systems (GIS), moving objects, spatial data sources, KML, PostGIS.

1. INTRODUCTION

This work extends (FERREIRA *et al.*, 2012), in two ways. First, the strategy designed only for KML files is generalized to deal with different kinds of spatial data sources, including database systems such as PostGIS. Second, we propose a new software architecture based on this extended strategy. To prove such strategy and architecture, we build a prototype and try out it with animal tracking and car movement data from different sources.

The recent technological advances in geospatial data collection, such as Earth observation and GPS satellites, mobile computing, and sensor networks, have motivated new applications that handle spatiotemporal information. Some examples are location-based systems, natural disaster and environmental change monitoring. To support these applications, there is a growing demand for geographical information systems (GIS) that deal with such information.

Since the beginning of the 2000s, the GIS community has made a serious effort towards spatial data interoperability. The International Organization for Standardization (ISO) and the Open Geospatial Consortium (OGC) have proposed standards to represent and store spatial information in data files and database systems. Geography Markup Language (GML) (OGC, 2007) and Keyhole Markup Language (KML) (OGC, 2008) are examples of file formats proposed by OGC for spatial data interchange. Many agencies and institutions throughout the world have distributed their spatial data using these formats. Spatial extensions of traditional Database Management Systems (DBMS), such as PostGIS and Oracle Spatial, deal with spatial information in compliance with the OGC Simple Feature Access (SFA) specification (OGC, 2006a) (OGC, 2006b).

The compliance with ISO and OGC standards has assured a high degree of spatial data interoperability. Many GIS tools and libraries are able to access spatial data files and databases that follow these standards. Standards are

useful to promote spatial data interoperability. However, few results have been achieved regarding spatiotemporal data interoperability.

Moving object is a well-known category of spatiotemporal data. They are objects whose spatial positions or extents change continuously over time (ERWIG *et al.*, 1999). Examples of moving objects are cars, aircraft, ships, mobile phone users, polar bears, hurricanes, forest fires, and oil spills on the sea. Although the concept of a moving object is well-established in GIS science, there is not a standard way to represent it in data files or database systems. Each data producer adopts its own format to store moving objects. A particular format specifies the way to encode information and how it is organized.

This work focuses on this class of spatiotemporal data. It proposes an interoperable strategy to translate spatial and temporal information stored in different data sources into moving object trajectories for further analyses. The proposed approach is based on the processing of an additional metadata file that describes *how* moving objects are stored in a particular data source. It is an XML file that must be compliant with a schema proposed in this paper. Grounded on this strategy, we have built a new software module to deal with and analyze moving objects in a geographical library called TerraLib (CÂMARA *et al.*, 2008).

2. RELATED WORK

Erwig *et al.* (1999) propose a model, called *Moving Object Model*, which defines an algebra to deal with moving objects. This algebra specifies three main data types, *moving points*, *moving lines* and *moving regions*, and a set of operations over them, such as *trajectory* and *distance*. This work is based on this algebra.

Fig. 1 (a) and (b) shows the tracking of an animal and the evolution of a deforested region. The former is an example of a moving point because the animal position changes over time. The latter is a moving region, since the object extent evolves over time.

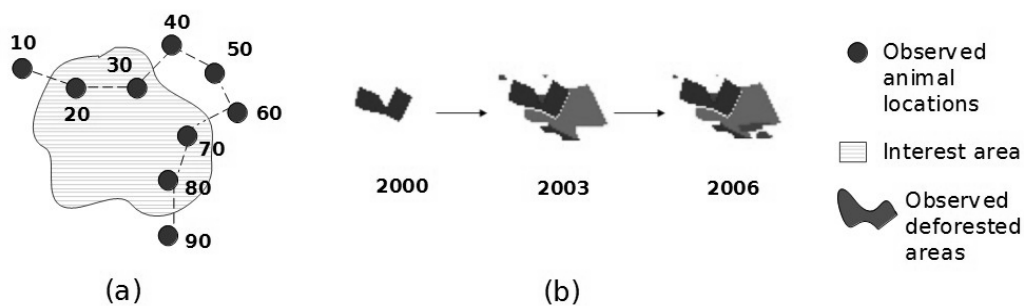


Fig. 1 - Examples of moving objects: (a) an animal tracking and (b) the evolution of a deforested region.

Although moving object spatial positions or extents change continuously over time, they are often represented by discrete observations. For instance, Fig. 1 (a) shows an animal tracking through an observation set. Each observation records a spatial position, represented by a point, and a time instant when the animal was at that position. Fig. 1 (b) presents the evolution of a deforested region through three observations. Each one contains the spatial extent of the deforested region, represented by a polygon, and the year when it was detected.

Trajectories are countable journeys associated to objects that are moving over time (SPACCAPIETRA et al., 2008). Different kinds of trajectories can be extracted from a moving object. For example, if an application is interested in studying the daily behavior of an animal, it can extract its trajectories by grouping its daily observations. In another case, the application might extract trajectories that group the animal observations by its intersection with some regions of interest.

Based on the algebra proposed by Erwig et al. (ERWIG et al., 1999), there are two main initiatives of Moving Object Database (MOD) systems, *SECONDO* (GUTING and SCHNEIDER, 2005) and *Hermes* (PELEKIS et al., 2008). Both extend the SQL type system with data types to represent moving objects, such as *moving point* and *moving region*, and a set of functions to deal with them. *SECONDO* is an extensible database system prototype designed at the

FernUniversität in Hagen. Hermes is a MOD engine that has been implemented as an Oracle data cartridge.

ISO defines a conceptual model called *Moving Feature Model* for features whose geometries move over time as a rigid body (ISO, 2008). It supports changes of location, translation and rotation, but not deformation of a feature. The *Moving Object Model* is broader than the *Moving Feature Model* because it supports geometry deformation over time. By dealing with geometry deformations, the model can cope with a class of environmental problems, like deforested region evolution show in Fig. 1 (b), where entity geometries move and deform over time.

3. THE PROBLEM

Most existing spatial file formats (e.g. KML and GML) and database systems (e.g. PostGIS) do not provide data types or structures to represent moving objects. They represent spatial and temporal information using structures and types predefined in ISO and OGC specifications. However, in these specifications, there is nothing about moving object representation in data files or database systems. Each data producer adopts its own format to do it. Therefore, this work addresses the problem: *how to translate spatial and temporal information stored in different data sources into moving object trajectories for further analyses?*

To illustrate this problem, let us consider two real examples of data sources that contain spatial and temporal information related to moving objects: a KML file and a PostGIS database.

3.1. Moving Objects in KML files

KML stands for Keyhole Markup Language and is an OGC standard for encoding and transporting representations of geographic data, mainly for data display in an Earth browser. It is an XML file that follows a predefined XML

schema. Such schema describes the grammar which KML file instances must be compliant with. All components of the KML schema are defined in the namespace with the identifier "http://www.opengis.net/kml/2.2".

The KML Schema defines an element called `kml::PlacemarkType` to represent spatial objects and time stamps associated to them. Spatial objects are represented by five types: `kml:MultiGeometryType`, `kml:PointType`, `kml:LineStringType`, `kml:LinearRingType` and `kml:PolygonType`. It defines two types for time information: `kml:TimeStampType` and `kml:TimeSpanType`.

The first example is a KML file generated by a project that monitors sea elephants in the Antarctica (INPE, 2012). This file contains observations of eight animals during three years. Each observation has an animal location at a specific time and is represented by a `kml::PlacemarkType` element. The animal location is represented by `kml::PointType` and its associated time by `kml::TimeStampType`.

Although KML is used to describe journeys, there is not a predefined type in its schema that associates spatial and temporal elements to a same trajectory. There is nothing to indicate what `kml::PlacemarkType` elements must be grouped as the same moving object trajectory. In this example, the KML file uses a `kml::FolderType` element to group all observations of the same animal. However, KML files generated by other producer can use different elements to do it.

This file also contains visual style elements to describe how the data should be visualized. Fig. 2 shows the display of this KML file in the Google Earth, where the red lines represent the sea elephant trajectories.

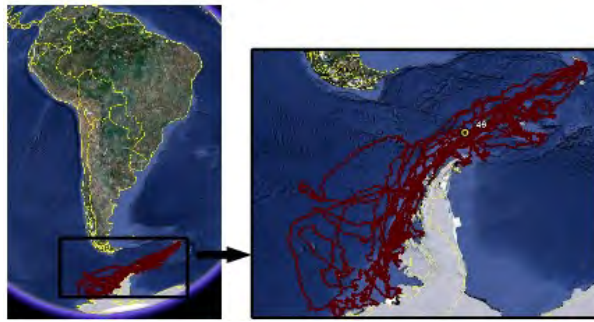


Fig. 2 - Trajectories of sea elephants: display of the KML in the Google Earth software.

3.2. Moving Objects in a PostGIS database

PostGIS extends the PostgreSQL, an open source object-relational database system, to deal with geographic objects. It is compliant with the OGC Simple Feature Access (SFA) specification (OGC, 2006a) (OGC, 2006b). It provides a set of data types to represent geometries, such as `st_point` and `st_polygon`, and of functions to handle these types, such as `st_distance` and `st_intersection`. These types and functions come from the OGC geometry model. For temporal information, PostgreSQL supports the full set of Structured Query Language (SQL) date and time types, such as `timestamp`, `interval`, `date` and `time`.

The SFA specification uses the term feature tables to refer to tables that have at least a spatial attribute, stored in a column whose domain is a geometry type. It proposes two metadata tables: `geometry_columns` and `spatial_ref_sys`. The `spatial_ref_sys` table holds the numeric identifications and textual descriptions of coordinate systems used in the spatial database. The `geometry_columns` table registers the available feature tables in the database and metadata about their geometry columns, such as their types and associated spatial reference systems identifications (`srid`).

The second example is a PostGIS database that has observations of moving cars in a city. Fig. 3 shows the trajectories of three cars during a day, where

each point represents a car location at a specific time. All observations of all cars are stored in a feature table, called `car_trajectories`, that has three columns: (1) `car_id`: to store the car identities; (2) `location`: to store the car spatial locations (`st_point` type); and (3) `date_time`: to store the temporal information (`timestamp` type).

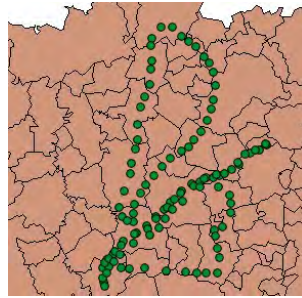


Fig. 3 - Trajectories of three cars in a city during a day.

The `car_trajectories` is a feature table and so the metadata about its geometry column is registered in the `geometry_columns` table. However, there is not a metadata in this database that indicates *how* to translate the spatial and temporal information of the `car_trajectories` table into moving object trajectories.

3.3. Analyzing Moving Objects

Most GIS tools can access and display geometries and their associated times from PostGIS databases and KML files. Some of them, such as Google Earth, can automatically configure timelines and generates animations over time. However, they are not able to analyze them as moving object trajectories. They cannot answer questions like: (1) *Where was object o_1 at time t_5 ?* (2) *When did object o_1 enter a specific region r_{10} and how long did it stay in this region?* (3) *When and where did objects o_1 and o_2 meet each other (considering a meeting when the distance between two objects is less than 2 meters)?* (4) *Where and when was there a spatiotemporal cluster of objects?*

This requires a more specialized tool that is able to: (1) translate geometry objects associated to time stamps stored in data sources into data structures that represent moving object trajectories, and (2) analyze trajectories, by providing functions over its data structures that can answer questions like the ones presented above. To meet these requirements, we are developing a new software module in a geographical library called TerraLib (CÂMARA et al., 2008). Its architecture is described at follow.

4. SOFTWARE ARCHITECTURE

This section describes the architecture of a new software module for moving object analysis, built in a geographical library called TerraLib. TerraLib is a C++ software library base to build geographical information systems. It is open source and is developed by the National Institute for Space Research (INPE) (CÂMARA et al., 2008).

This new module is composed of three other ones, ST (SpatioTemporal), STLoader and DataAccess, as shown in Fig. 4. The ST module contains data structures and functions to represent and analyze moving objects. It provides functions to calculate the distance between two moving objects and the intersection between a moving object and a region of interest. The distance operation results in a time series that maps each time to the distance between the objects at that time. The intersection operation results in patches or trajectories of a moving object that intersect a region of interest, as shown in Fig. 5. In this figure, each trajectory represents a patch when the object was inside the region of interest. Using the ST module functions, a user can answer questions like the four ones presented in Section 3.3.

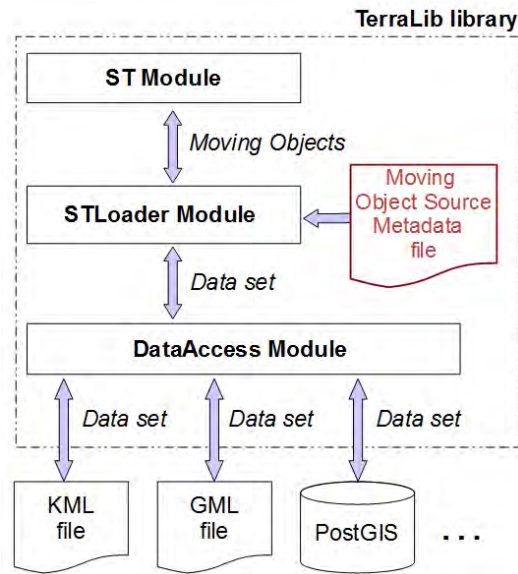


Fig. 4 - Software architecture.

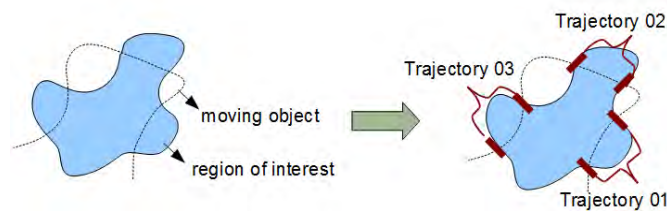


Fig. 5 - Intersection between a moving object and a region of interest.

The DataAccess module is in charge of accessing data sets from different sources, such as KML and GML files as well as PostGIS databases. Each source stores spatial and temporal information using particular predefined structures. A point is stored in a `km1::PointType` element in KML files and in a `st_point` type in PostGIS databases. So, this module has to know the particularities of each source to load its data sets.

The STLoader module is responsible for translating the data sets loaded by the DataAccess into moving object structures of the ST module. To do this, it needs extra information about *how* the sources represent moving objects. Let us consider the PostGIS database presented in Section 3.2. To load its moving cars, this module has to know that the `car_trajectories` table contains

moving objects. Besides that, it needs to know that its column `car_id` stores the car identities, `location` stores the car locations and `date_time` contains the temporal information. To load moving animals from the KML file described in Section 3.1, this module has to know that all observations of each animal are grouped in a `km1::FolderType` element.

Therefore, this module requires an additional metadata file, called *moving object source metadata*, which contains this necessary extra information.

5. MOVING OBJECTS SOURCE METADATA

The *moving object source metadata* is an XML file. XML stands for eXtensible Markup Language and is a markup language designed to transport and store structured data. It is a World Wide Web Consortium (W3C) recommendation and has been widely used to carry and share data mainly in the Web environment (BRAY et al., 2008). An XML file is structured through user-defined tags and can be described by a XML Schema. The purpose of an XML Schema is to define the legal building blocks of an XML document in terms of elements and attributes that can appear in an XML file. The XML Schema language is called XML Schema Definition (XSD).

Moving object source metadata files must be compliant with the XML Schema proposed in this section. This schema is shown in Fig. 6. It defines seven complex elements: `MovingObjectType`, `DataSourceInfoType`, `DataSourceParamsType`, `MovingObjectInfoType`, `IdInfoType`, `SpatialInfoType` and `TemporalInfoType`.

`MovingObjectType` is the root element. It encloses all the other elements that contain metadata about data sources and their moving objects. Information about each data source is described by the `DataSourceInfoType` element. It holds the data source name, its type and its access parameters. In this first version, the metadata file supports two types of data sources, KML and POSTGIS.

```

<xs:complexType name="MovingObjectSourceType">
  <xs:sequence>
    <xs:element name="DataSourceInfo"
      type="DataSourceInfoType"
      use="required"/>
    <xs:element name="MovingObjectInfo"
      type="MovingObjectInfoType"
      use="required"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="DataSourceInfoType">
  <xs:sequence>
    <xs:element name="name"
      type="xs:string"
      use="required"/>
    <xs:element name="type" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="KML"/>
          <xs:enumeration value="POSTGIS"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="Params"
      type="DataSourceParamsType"
      use="required"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="DataSourceParamsType">
  <xs:sequence>
    <xs:element name="keyValuePair"
      type="xs:string"
      use="required"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="MovingObjectInfoType">
  <xs:sequence>
    <xs:element name="containerType"
      type="xs:string"
      use="required"/>
    <xs:element name="containerName"
      type="xs:string"
      use="required"/>
    <xs:element name="IdInfo"
      type="IdInfoType"
      use="optional"/>
    <xs:element name="SpatialInfo"
      type="SpatialInfoType"
      use="required"/>
    <xs:element name="TemporalInfo"
      type="TemporalInfoType"
      use="required"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="IdInfoType">
  <xs:sequence>
    <xs:element name="name"
      type="xs:string"
      use="required"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SpatialInfoType">
  <xs:sequence>
    <xs:element name="name"
      type="xs:string"
      use="required"/>
    <xs:element name="srid"
      type="xs:integer"
      use="optional"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="TemporalInfoType">
  <xs:sequence>
    <xs:element name="name"
      type="xs:string"
      use="required"/>
    <xs:element name="pattern"
      type="xs:string"
      use="optional"/>
    <xs:element name="resolution"
      use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="SECOND"/>
          <xs:enumeration value="MINUTE"/>
          <xs:enumeration value="HOUR"/>
          <xs:enumeration value="DAY"/>
          <xs:enumeration value="MONTH"/>
          <xs:enumeration value="YEAR"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:schema>

```

Fig. 6 - The Moving Object Source Metadata file schema.

The access parameters are described by the DataSourceParamsType element. Since each type of data source requires a specific set of access parameters, this element is composed of key-value pairs instead of predefined elements. To access a PostGIS database, a user opens a connection that requires, at least, the database name, the server host name and its available port, the user name and its password. Otherwise, to access and open a KML file, a system only needs its path and name. A list of possible access parameters for each data

source type is available in the TerraLib documentation available at www.terralib.org.

`MovingObjectInfoType` element carries information about the containers in the data sources that hold moving object observations. It includes: (1) the container type and name (`containerType` and `containerName` elements); (2) where the object identities are stored (`IdInfoType` type); (3) where the spatial and temporal information is stored (`SpatialInfoType` and `TemporalInfoType` types). The container that holds the moving car observations (Section 3.2) is the table `car_trajectories`. The identity of each car is stored in the column `car_id`. The columns `location` and `date_time` store the spatial and temporal information of each observation.

The `IdInfoType` element describes where the object identities are stored. The `SpatialInfoType` element describes where the spatial information is stored and its Spatial Reference System Identification (SRID). SRID is a unique number used to identify projected and local spatial coordinate system definitions. In the metadata file, the `srid` is optional since it can be already registered in the data source. A PostGIS database holds the `srid` of its feature tables in the `geometry_columns` table.

The `TemporalInfoType` element indicates where the temporal information is stored as well as its pattern and temporal resolution. Temporal pattern refers to the format of a textual representation of a date and time. For example, the text “01-03-2008” is ambiguous; it can represent the first day of March in 2008 or the third day of January in 2008. So, we have to inform what pattern it follows in order to understand its right meaning. ISO 8601:2004 (ISO, 2004) proposes some date and time format representations, such as DD-MM-YYYY or MM-DD-YYYY, and this work adopts them. The temporal pattern information is optional. It is only necessary when the temporal data is of a textual type.

Temporal resolution refers to the time granularity which must be considered to deal with temporal information. Each deforested region observation (shown in Fig. 1 (b)) can have a complete date associated to it (a day, a month and a year), such as '01-01-2003'. Nevertheless, the measurement of deforested regions is done yearly and so only the year must be considered in this date. In other words, the time resolution associated to it is YEAR. The possible time resolutions are: YEAR, MONTH, WEEK, DAY, HOUR, MINUTE and SECOND.

6. EXAMPLES

This section presents the moving object source metadata files related to the two examples presented in Section 3. These files contain necessary information to translate the spatial and temporal information in the KML file and in the PostGIS database into moving animals and moving cars, respectively. They follow the schema described in the previous section.

6.1. Moving Animals in the KML file

Fig. 7 (a) presents the moving object metadata file related to the KML file described in Section 3.1. The DataSourceInfo element contains the data source name (sea_elephants); its type (KML) and its access parameters (NAME=c://sea_elephants.kml). To access and open a KML file, a system needs only its name and path. So, we inform only the access parameter NAME.

In this file, there are two MovingObjectInfo elements that describe information about two containers of moving animal observations. The first container is a folder (containerType is kml::FolderType) called 40: locations (containerName). The second one is also a folder (containerType is kml::FolderType) called 41: locations (containerName). Both folders hold moving animals observations using the following KML elements: (1) kml::Placemark::Name to store the animal identities (IdInfo); (2) kml::Placemark::Point to store the animal locations (SpatialInfo); and (3) kml::Placemark::TimeStamp to store the temporal information associated to

each location (TemporalInfo). The srid of the locations is 4326 that refers to the reference coordinate system WGS84. And, its temporal resolution is SECOND.

6.2. Moving Cars in the PostGIS database

Fig. 7 (b) presents the moving object metadata file related to the PostGIS database described in Section 3.2. The DataSourceInfo element contains the data source name (cars), its type (POSTGIS) and its access parameters. These parameters contain necessary information to open a connection to a PostGIS database: the database name (NAME=stdatabase), the server host name (HOST=localhost), its available port (PORT=5432) and the user name (USER=postgres).

Information about how the moving cars are stored in this database is in the MovingObjectInfo element. The moving car observations are stored in a table (containerType is Table) called car_trajectories (containerName). The car identities are stored in a column called car_id (IdInfo). The car spatial locations are stored in a column called location (SpatialInfo) and their associated times in a column called date_time (TemporalInfo). We do not need to inform the srid in this file. It comes from the geometry_columns where the table car_trajectories is registered.

<pre> <?xml version="1.0" encoding="UTF-8" standalone="no"?> <MovingObjectSource xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.terralib.org/schemas/movingobjectsource" xsd:schemaLocation= "http://www.terralib.org/schemas/movingobjectsource movingobjectsource.xsd"> <DataSourceInfo> <name>sea_elephants</name> <type>KML</type> <Params> <keyValuePair>NAME=C://sea_elephants.kml </keyValuePair> </Params> </DataSourceInfo> <MovingObjectInfo> <containerType>kml::FolderType</containerType> <containerName>40: locations</containerName> <IdInfo> <name>kml::Placemark::Name</name> </IdInfo> <SpatialInfo> <name>kml::Placemark::Point</name> <srid>4326</srid> </SpatialInfo> <TemporalInfo> <name>kml::Placemark::TimeStamp</name> <resolution>SECOND</resolution> </TemporalInfo> </MovingObjectInfo> <MovingObjectInfo> <containerType>kml::FolderType</containerType> <containerName>41: locations</containerName> <IdInfo> <name>kml::Placemark::Name</name> </IdInfo> <SpatialInfo> <name>kml::Placemark::Point</name> <srid>4326</srid> </SpatialInfo> <TemporalInfo> <name>kml::Placemark::TimeStamp</name> <resolution>SECOND</resolution> </TemporalInfo> </MovingObjectInfo> </MovingObjectSource> </pre> <p style="text-align: center;">(a)</p>	<pre> <?xml version="1.0" encoding="UTF-8" standalone="no"?> <MovingObjectSource xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.terralib.org/schemas/movingobjectsource" xsd:schemaLocation= "http://www.terralib.org/schemas/movingobjectsource movingobjectsource.xsd"> <DataSourceInfo> <name>cars</name> <type>POSTGIS</type> <Params> <keyValuePair>NAME=stdatabase </keyValuePair> <keyValuePair>HOST=localhost </keyValuePair> <keyValuePair>PORT=5432 </keyValuePair> <keyValuePair>USER=postgres </keyValuePair> </Params> </DataSourceInfo> <MovingObjectInfo> <containerType>Table</containerType> <containerName>car_trajectories</containerName> <IdInfo> <name>car_id</name> </IdInfo> <SpatialInfo> <name>location </name> </SpatialInfo> <TemporalInfo> <name>date_time </name> <resolution>SECOND</resolution> </TemporalInfo> </MovingObjectInfo> </MovingObjectSource> </pre> <p style="text-align: center;">(b)</p>
--	---

Fig. 7 - The moving object metadata files: (a) related to the KML file and its moving animals (Section 3.1); (b) related to the PostGIS database and its moving cars (Section 3.2).

7. PROTOTYPE

We have built a new software module to deal with and analyze moving objects in the geographical library TerraLib. This module is based on the software architecture and strategy proposed in this paper. Besides that, we have adapted the open GIS TerraView to display and analyze moving objects, using this new module. TerraView is a geographical application built utilizing the TerraLib library (INPE, 2012). It is open source and developed by INPE.

Fig. 8 shows TerraView displaying the trajectories of two sea elephants (blue and yellow lines at the bottom) and the distance between both. The distance operation results in a time series (right side of the figure) that maps each time to the distance between both at that time. TerraView has loaded these two trajectories from the KML shown in Fig. 2, using the *moving object source metadata* file presented in Fig. 7 (a). It can also display them through an animation over time.

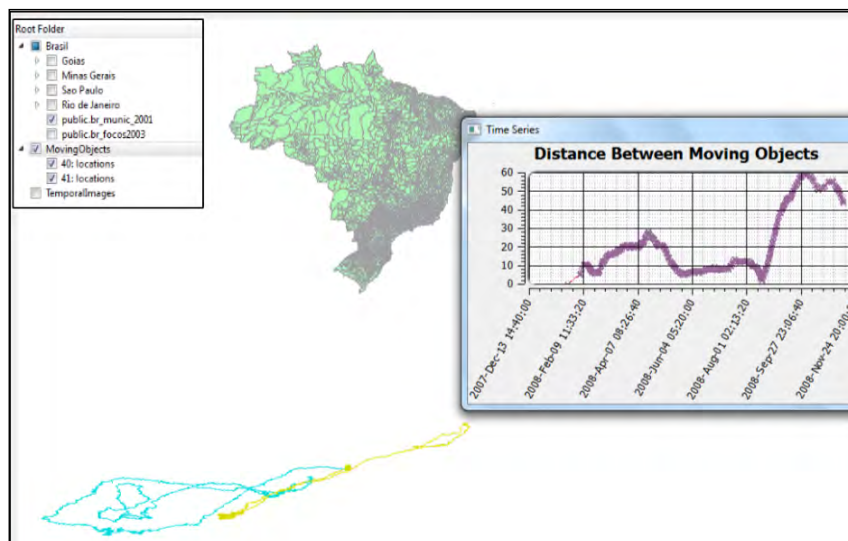


Fig. 8 - TerraView: displaying and analyzing sea elephant trajectories.

We have built this module using three open source C++ software libraries: Xerces-C++, OGR and libpq. Xerces-C++ (<http://xerces.apache.org/xerces-c/>) is able to read and write XML data, checking its compliance with predefined schemas. It is used to read the moving object source metadata files that are XML files. OGR provides read (and sometimes write) access to a variety of geographical vector file formats, including KML files (<http://www.gdal.org/ogr/>). We use the OGR LIBKML Driver to read KML files (http://www.gdal.org/ogr/drv_libkml.html). To access PostGIS databases, we use the libpq library (<http://www.postgresql.org/docs/8.2/static/libpq.html>).

8. FINAL REMARKS

The proposed approach consists in loading spatial and temporal information from data sources as it is and, afterwards, translating it into moving objects trajectories. To do this, it uses an additional metadata file that describes *how* moving objects are stored in a particular data source. This translation is essential to analyze the original information as moving object trajectories. To answer the question “*when and where did objects o_1 and o_2 meet each other (considering a meeting when the distance between two objects is less than 2 meters)?*”, we need to structure the original data as moving object trajectories.

This strategy has two main advantages. The first one is that no change in the original data sources is required. It loads the original data as it is and uses the metadata file to know how to translate it into moving objects. This feature is particularly interesting when dealing with database servers and the final application do not have permission to change them.

The second advantage is that it can be easily extended to other data sources. In this paper, we show a prototype working with KML files and PostGIS databases. However, we can easily extend it to other kinds of data sources, such as GML or Oracle Spatial. To do it, we have to: (1) add the new types of data sources in the moving object source metadata file schema, including them in the element type of the `DataSourceInfoType` type (Fig. 6); and (2) build a new software piece in the `DataAccess` module that is able to load spatial and temporal information from these new data sources.

This proposal allows for dealing with moving objects data using common GIS spatial files and DBMS spatial extensions. In this perspective, this work advances towards a new generation of GIS that deals with spatiotemporal data.

References

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. M.; MALER, E.; YERGEAU, F. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C recommendation. Report. W3C, 2008.

CÂMARA, G.; VINHAS, L.; FERREIRA, K.; QUEIROZ, G.; SOUZA, R. C.; MONTEIRO, A. M. V.; CARVALHO, M. T.; CASANOVA, M. A.; FREITAS, U. M. TerraLib: An Open Source GIS Library for Large-scale Environmental and Socio-economic Applications. Open Source Approaches to Spatial Data Handling, Berlin, Springer-Verlag, 2008.

ERWIG, M.; GUTING, R. H.; SCHNEIDER, M.; VAZIRGIANNIS, M. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. Geoinformatica. v. 3, p. 265-291, 1999.

FERREIRA, K. R.; VINHAS, L.; MONTEIRO, A. M. V.; CÂMARA, G. Moving Objects and KML Files. In: Proceedings of the 28th IEEE International Conference on Data Engineering (ICDE 2012) Workshop on Spatio Temporal data Integration and Retrieval. Washington D.C., USA, 2012.

GUTING, R. H.; SCHNEIDER, M. Moving Objects Databases. San Francisco, Morgan Kaufmann, 2005.

INPE. Projeto MEOP: INPE, 2012. Available at: <<http://www.inpe.br/crs/pan/pesquisas/telemetria.php>>. Access data: 12/07/2012.

INPE. TerraView software. São José dos Campos, SP: INPE, 2012. Available at: <http://www.dpi.inpe.br/terraview_eng/index.php>. Access data: 12/07/2012.

INTERNATIONAL STANDARD ORGANIZATION (ISO). ISO 8601:2004: Data elements and interchange formats - Representation of dates and times. Report. Geneva, Switzerland, 2004.

INTERNATIONAL STANDARD ORGANIZATION (ISO). ISO 19141:2008: Geographic information - Schema for moving features. Report. Geneva, Switzerland, 2008.

OPEN GEOSPATIAL CONSORTIUM (OGC): OpenGIS Implementation Specification for Geographic Information – Simple Feature Access - Part 1: Common architecture. Reference number: OGC 06-103r3. Version: 1.2.0. Report. Available at <<http://www.opengeospatial.org>>. 2006a.

OPEN GEOSPATIAL CONSORTIUM (OGC): OpenGIS Implementation Specification for Geographic Information – Simple Feature Access - Part 2: SQL option. Reference

number: OGC 06-104r3. Version: 1.2.0. Report. Available at <<http://www.opengeospatial.org>>. 2006b.

OPEN GEOSPATIAL CONSORTIUM (OGC). OpenGIS Geography Markup Language (GML) Encoding Standard. Reference number: OGC 07-036. Version: 3.2.1. Report. Available at: <<http://www.opengeospatial.org>>. 2007.

OPEN GEOSPATIAL CONSORTIUM (OGC). OGC KML. Reference number: OGC 07-147r2. Version: 2.2.0. Report. Available at: <<http://www.opengeospatial.org>>. 2008.

PELEKIS, N.; FRENTZOS, E.; GIATRAKOS, N.; THEODORIDIS, Y. HERMES: Aggregative LBS via a Trajectory DB Engine. In: Proceedings of the ACM SIGMOD' 08 Conference. Vancouver, BC, Canada. 2008.

SPACCAPIETRA, S.; PARENT, C.; DAMIANI, M.; MACEDO, J. A. F.; PORTO, F.; VANGENOT, C. A conceptual view on trajectories. Data & Knowledge Engineering. v. 65, p. 126-146, 2008.