DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TPiet-QL: A Spatio-Temporal Query Language for OLAP

January 2013

Pablo N. Bisceglia
pbisceglia@gmail.com

Director
Alejandro Vaisman
avaisman@ulb.ac.be

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Abstract

En los últimos tiempos se han desarrollado aplicaciones, denominadas SOLAP (por Spatial OLAP), que integran Sistemas de Información Geográfica (GIS) y de Procesamiento Analítico On-Line(OLAP), con el objetivo de explorar y analizar datos espaciales y alfanuméricos en forma conjunta. En la práctica, sin embargo, la información espacial y no espacial está sujeta a cambios. Por ejemplo, las aplicaciones de uso catastral requieren la capacidad de manejar situaciones en las que las parcelas se fusionan, se dividen o se modifican. Estos tipos de cambio se denominan *discretos*, para distinguirlos de aquellos que denotan movimiento *continuo* (ej: un auto que transita una autopista). Soportar estos tipos de cambio requiere extender SOLAP de modo de soporte aplicaciones temporales. En esta tesis presentaremos el diseño e implementación de un lenguaje de consulta espacio-temporal llamado TPiet-QL, que soporta cambios discretos mediante una extensión de Piet-QL, un lenguaje de consulta (no temporal) para SOLAP. TPiet-QL permite expresar consultas integradas GIS-OLAP en un escenario en el que los objetos espaciales se modifican a través del tiempo.

Keywords: GIS, OLAP, SOLAP, Piet-QL.

Abstract

In recent years, applications integrating Geographic Information Systems (GIS) and On-Line Analytical Processing (OLAP) have emerged, aimed at exploring and analyzing spatial data. These applications have been called SOLAP (Spatial OLAP). In real-world SOLAP applications, spatial and non-spatial data are subject to changes. For example, land use or cadastral applications require the ability to handle situations where parcels are merged, split, or updated. These kinds of changes have been called discrete, to distinguish them from other kind of spatio-temporal data, denoted continuous motion (e.g., a car moving in a highway). Therefore, SOLAP needs to be extended in order to support these kinds of applications. In this thesis we present the design and implementation of a spatio-temporal query language called TPiet-QL, that supports discrete changes, extending Piet-QL, a query language for (non-temporal) SOLAP. TPiet-QL allows expressing integrated GIS-OLAP queries in an scenario where spatial objects change across time.

Keywords: GIS, OLAP, SOLAP, Piet-QL.

Acknowledgements

I want to thank God for giving me a family that showed me the importance of compromising with goals and making efforts in life. Thanks to my wife Valeria, for encouraging me to continue studying and her unconditional love. To my daughter Malva, who always had her beautiful smile and patience while I was preparing this final work.

Related to my thesis activity, I want to thank Leticia Gómez for all the invaluable time, help and effort she dedicated to me during this period.

Finally, I want to remark my special thanks to my director Alejandro Vaisman for his work and dedication. I am sure that without his direction and advice it would have been impossible for me to finish this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In Geographic Information Systems (GIS), spatial information is stored in *thematic layers*. Spatial data are stored in suitable data structures, and associated attributes are usually stored in conventional relational databases. Spatial data in different layers can be mapped univocally to each other using a common frame of reference, like a coordinate system. Layers can be overlayed to obtain an integrated spatial view. On the other hand, OLAP (On-Line Analytical Processing)[13] provides a set of tools and algorithms that allow efficiently querying multidimensional data repositories called Data Warehouses. OLAP data is organized as a set of *dimensions* (organized as hierarchies) and *fact tables*, and can be perceived as a *data cube*, where each cell contains a measure or set of aggregated measures of interest. The problem of integrating OLAP-centric systems and GIS-centric systems, has been called SOLAP[4]. One of the models proposed for SOLAP is denoted Piet[9][1], a framework that integrates spatial, spatio-temporal, and non-spatial multidimensional data.

Spatial objects in thematic layers can be added, removed, split, merged, or their shape may change. Tryfona and Jensen[21] classify spatio-temporal applications according to the kind of changes, occurring in the spatial objects that these applications can support. They make a difference between objects with *continuous motion* (e.g., a car moving in a highway), objects with *discrete changes* (e.g, parcels changing boundaries), and objects combining *continuous motion and changing shapes*, for instance, a storm or a stain in a river.

The Piet framework and its corresponding query language Piet-QL assume that all objects in a layer remain unchanged across time. In this thesis we present a temporal extension to Piet-QL that supports *discrete changes*.

[1] A description and demo of Piet can be found at http://piet.exp.dc.uba.ar/piet.

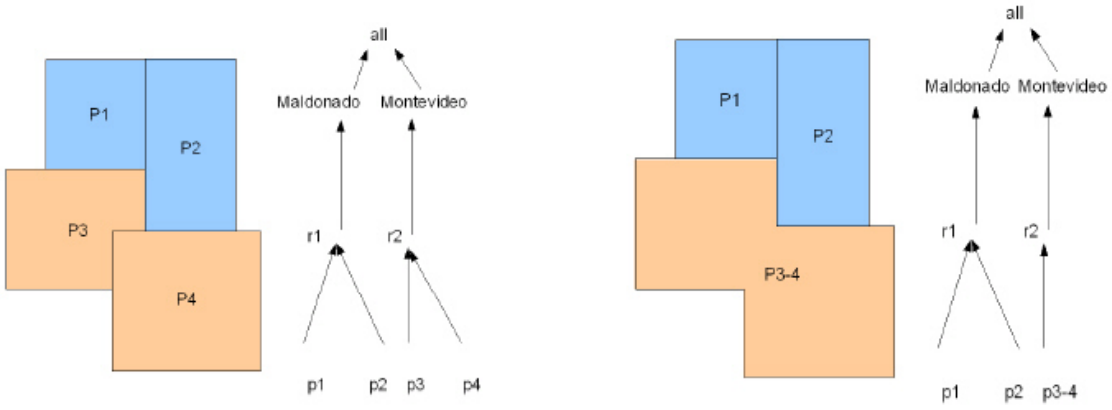


Figure 1.1: Initial situation(left): land partition and land dimension hierarchy; situation after merging P3 and P4 (right): changes in spatial objects and in the dimension hierarchy

1.1 Motivation and Contributions

A Motivating Example. Let us assume the following scenario about land property information. In Figure 1.1 (left) we show four land parcels, P_1 through P_4. They may be characterized, for example, by attributes like type of soil, depth of underground water availability, and owner. The parcels are subject to change in shape, area, and in the value of their attributes, and we consider them as composing a single GIS layer denoted L_{land}. There is also non-spatial information stored in a conventional data warehouse. In particular, a dimension hierarchy denoted *Land*, which stores information related to parcels. The bottom level of this dimension contains parcel identifiers (p_1 through p_4). These *IDs* aggregate over a level denoted *region*, which, in turn, aggregates over a *province* level. There is a mapping (not shown in the figure) defined between spatial objects in L_{land} and members of the bottom level (*parcelId*) of the dimension *Land*. In this case, the mapping is complete, but this is not mandatory. (The mapping associates each P_i object -a parcel- the corresponding dimension level member p_i). At a certain moment, the owner of P_3, acquires P_4. Thus, parcels P_3 and P_4 are merged into a single one P_{3-4}. Changes must also be performed at the data warehouse, meaning that elements p_3 and p_4 are deleted and p_{3-4} is added, along with the corresponding rollups to region r_2. A mapping between p_{3-4} and P_{3-4} is also defined. This is described on the right hand side of Figure 1.1. Some time later, other changes may occur, e.g., P_2 may grow if the owner decides to sell part of P_{3-4} to the owner of P_2. In a discrete changes scenario like this, we may want to know the history of P_{3-4}, the production of each existing parcel as of the year 2006, or to pose queries like "Production by year per square mile for each parcel of land, for the parcels in Montevideo". Answering these kinds of queries requires extending non-temporal SOLAP data models and query languages (like Piet-QL) with temporal capabilities.

The main contribution of this thesis consists of a formal definition for syntax and semantics of the temporal operators, predicates and functions that compose a temporal query language for SOLAP, denoted TPiet-QL. We also show how the topological predicates usually found in the GIS literature are modified to support temporal semantics, and discuss the language expressiveness.

1.2 Thesis Organization

This document is organized as follows. After an overview of related work (*Chapter 2*), we define the temporal data model (*Chapter 3*), and present the syntax and semantics of TPiet-QL in *Chapter 4*. In *Chapter 5*, implementation details of the software components are included and then we describe the datasets used in our case study and the transformation process developed to prepare this data, according to our data model (*Chapter 6*). We also provide several queries over the datasets of our case study in order to get an insight of the query language in *Chapter 7*, concluding in *Chapter 8*.

This thesis has been developed as part of the LACCIR[2] project R1210LAC004 - *Monitoring Protected Areas using an OLAP-enabled Spatio-Temporal Geographic Information System*[3]. The main results of this thesis have been previously published in [5] and have been presented in a technical session of the AMW2012[4].

[2]http://www.laccir.org
[3]http://www.fing.edu.uy/inco/proyectos/laccirPiet/
[4]http://www2.dcc.ufmg.br/eventos/amw2012/dokuwiki/doku.php

Chapter 2

Related Work

Rivest *et al.*[17] introduced the concept of SOLAP (standing for Spatial OLAP), a paradigm aimed at exploring spatial data by drilling on maps in a way analogous to what is performed in OLAP with tables and charts. They describe the desirable features and operators a SOLAP system should have.

Piet[9] is a formal model for SOLAP, where the integration between GIS and OLAP is materialized through a function that maps elements in the data warehouse to elements in the GIS layers. Piet comes equipped with a query language, Piet-QL [10], that supports the operators proposed by the Open Geospatial Consortium[1] for SQL, adding the necessary syntax to integrate OLAP operations through MDX[2]. Piet-QL is designed to support four basic kinds of queries: (a) GIS queries filtered using spatial conditions, like "Regions which contain cities crossed by rivers"; (b) OLAP queries filtered using OLAP conditions, like "Total sales of products in cities with sales higher than 5000 units"; (c) GIS queries filtered using OLAP conditions, like "Name of the cities with total sales higher that 5000 units"; (d) OLAP queries filtered by spatial conditions, like "Total sales in cities within 100 Km from Montevideo". Queries of type (c) and (d) characterize integrated GIS-OLAP queries. Filtering is materialized through a predicate denoted **IN**. To give an idea of how a Piet-QL query looks like, consider the following one: "Parcels crossed by the 'Uruguay' river, with sales greater than 5000 units".

```
SELECT GIS l.id
FROM land l, rivers lr
WHERE intersects(l,lr) AND lr.name = 'Uruguay'
AND l IN (SELECT CUBE filter([Land].[Land parcelId].Members,
          [Measures].[Parcel Sales] > 5000)
         FROM [Sales])
```

[1] http://www.opengeospatial.org

[2] MDX is a query language initially proposed by Microsoft as part of the OLEDB for OLAP specification, and later adopted as a standard by most OLAP vendors. See http://msdn2.microsoft.com/en-us/library/ms145506.aspx.

Here, *land* and *rivers* represent two thematic layers containing spatial objects (the parcel subdivision of a given region, and the rivers, respectively). The OLAP subquery (identified with the keyword `CUBE`) is linked to the outer query by the predicate `IN`, and returns a collection of identifiers of spatial objects.

The Spatio-Temporal Relational data Model (STRM), introduced by Tryfona and Hadzilacos[20], provides a set of constructs consisting in relations, layers, virtual layers, object classes, and constraints, all with spatial and temporal extent, on top of wellknown models. In this model, a layer is a set of geometric figures like points, lines, regions or combinations of them, with associated values. The authors also define a layer algebra, which, based on four operations over layers, provides a semantics to SOLAP. Many proposals study moving object databases. SECONDO[11] is an extensible DBMS platform, composed of three major components which can be used together or independently: (i) the kernel, which offers query processing over a set of implemented algebras, each offering some type constructors and operators, (ii) the optimizer, which implements the essential part of an SQL-like language, and (iii) the graphical user interface which is extensible by viewers for new data types. The Hermes system[15] provides the functionality needed for handling two-dimensional objects that change location, shape and size. Both proposals support continuous motion, therefore we do not extend on commenting them. Other spatio-temporal data models exist, like [23], [22] Pelekis *et al.*[16] gives a comprehensive overview of spatio-temporal models. In spite of their ability to handle spatio-temporal data, none of the models above (except Piet) are oriented towards addressing the problem of integrating GIS, OLAP and Moving Object data.

Chapter 3

Spatio-Temporal Piet

In the temporal extension to Piet, denoted TPiet, each tuple in a relation is timestamped with its validity interval. Time is introduced into Piet as a new sort (domain). Toman[19] showed the equivalence between abstract and concrete temporal databases. The former are point-based structures, independent from the database's actual implementation. The latter are efficient (interval-based) encodings of the abstract databases. The author also shows that there is an efficient translation from abstract to concrete temporal databases. Formally, given a set T, and "$<$" a discrete linear order without endpoints on T, the structure $T_P = (T; <)$ is a Point-based Temporal Domain, where the elements in the carrier of T model the individual time instants, and the linear order "$<$" models the time sequence. We consider the set T to be $\mathbb{N}$, standing for the natural numbers. For clarity of presentation, we work with point-based temporal domains, although we use interval-based domains to implement our ideas. In temporal databases, the concepts of valid and transaction times refer to the instants when data are valid in the real world, and when data are recorded in the database, respectively [18]. We assume valid time support in this thesis. Also, as usual in temporal databases, a distinguished variable *Now* represents the (moving) current time instant. We also need the following definition:

Definition 1. *(Lifespan)* The *lifespan* of a GIS layer L, denoted *lifespan(L)*, is the collection of all the time instants where the layer is valid. The *lifespan* of a set of layers $\mathcal{L}$, denoted *lifespan($\mathcal{L}$)* is the union of the lifespans of all the layers in $\mathcal{L}$.

We assume that no structural changes occur at the GIS or at the data warehouses, meaning that a layer containing polygons at its creation instant will contain polygons throughout its lifespan. This assumption also implies that an OLAP dimension schema remains unchanged throughout its lifespan, i.e., its attributes and levels do not change. However, the spatial objects in the layers and the members of the OLAP dimensions can change, and these changes may also impact the mapping functions.

3.1 Formal Data Model

We have the following sets: a set of layer names **L**, a set of attribute and dimension level names **A**; a set **D** of OLAP dimension names, and a set **G** of geometry names. Each element a of **A** has an associated set of values *dom(a)*. We assume that **G** contains the following elements (geometries): point, node, line, polyline, and polygon. Each geometry G of **G** has an associated domain *dom(G)*, composed of a set of geometry identifiers g_{id}. In other words, g_{id} are identifiers of geometry instances (for example, polylines or polygons). Also, each g_{id} is associated with a list of coordinates that defines the geometric element. We denote this list the extension of g_{id}, *ext(g_{id})*.

Definition 2 *(Temporal GIS-OLAP Dimension Schema)*
There is a set of layers $L_1, \ldots, L_k \in \mathbf{L}$ such that for each L_i there is an associated kind of geometry $G_i \in \mathbf{G}$ (e.g. a layer can only contain points, polylines, or polygons, but it cannot contain combinations of them). We denote H the (total) function defining this mapping, with signature $\mathbf{L} \rightarrow \mathbf{G}$. There is also a set of dimension schemas $\mathcal{D}$ defined as in Hurtado and Mendelzon[12] where each dimension D $\in \mathcal{D}$ is a tuple of the form $\langle$*dname*,A,$\leq\rangle$, such that *dname* $\in \mathbf{D}$, A $\in \mathbf{A}$, is a set of dimension levels, and $\leq$ is a partial order between levels. A set $\mathcal{A}$ of *partial* functions *Att* with signature $\mathbf{A} \times \mathbf{D} \rightarrow \mathbf{L}$, maps attributes in OLAP dimensions to layers (see **Definition 3**). We denote μ a dimension level in a standard OLAP Time dimension, defined according with the granularity of the problem. Elements in μ in the temporal domain.

A *temporal GIS-OLAP dimension schema* TG_{sch} is the tuple $\langle H,\mathcal{A},\mathcal{D},\mu\rangle$, where H, $\mathcal{A}$ and $\mathcal{D}$ satisfy the following conditions: (a) A layer is created when the first object is added to it, (b) H is constant throughout the lifespan of the GIS, (c) For each layer $L \in \mathbf{L}$, the function *Att* is defined only in $lifespan(L)$, (d) The functions *Att* $\in \mathcal{A}$ do not change with time, i.e., Att_1 *(parcelId, Land)* will always return L_{land}, (e) The schema of the dimensions in $\mathcal{D}$ is constant during the lifespan of the GIS.

According to this definition the lifespan of a layer is determined by the lifespan of its objects. Associated with a dimension schema, we have a dimension instance.

Definition 3 *(Temporal GIS-OLAP Dimension Instance)* Let TG_{sch} be a Temporal GIS-OLAP dimension schema. A *timestamped temporal GIS-OLAP dimension instance is a tuple* $\langle \mathrm{TG}_{sch}, \mathcal{I}^t, A^t_{inst}, D^t_{inst}\rangle$, where: (a) $\mathcal{I}^t$ is a set of relations $r^t_{L_i}$ such that each tuple $\langle g_i, ext(g_i,t)\rangle$ in $r^t_{L_i}$, represents the existence of an object g_i (and its extension) in L_i at the instant t. (b) Associated with each function *Att* such as *Att*(A,D)= L, there is a set of functions $\alpha \in \mathcal{A}^t_{inst}$ with signature $L \times D \times A \times dom(A) \times dom(\mu) \rightarrow dom(G)$, where $A \in \mathbf{A}$, G is such that $H(L) = G$ in TG_{sch} (c)D^t_{inst} is a set of dimension instances, one for each dimension schema $D \in \mathcal{D}$ in TG_{sch}.

Definitions 2 and 3 assume that the attributes associated with each GIS layer are invariant, i.e., if the polygons representing parcels in L_{land} have the same attributes throughout

their lifespan, which is a reasonable assumption in practice. For instance, a designer may define that a parcel has attributes associated with land use or land ownership, and in case part of this information is missing, this will occur at the instance level, but the attributes will still be present in the schema of the layer.

Example 1: Consider two layers L_{land} and L_a, representing parcels and airports, respectively. Then, and example of the set $\mathcal{I}^t$ in an interval-based temporal database is:

$\mathcal{I}^t = \{r^t_{L_{land}} = \{\langle P1, ext(P1),$ {[0,5], [10,20], [40,*Now*]}$\rangle$, ...}, $r^t_{L_a} = \{\langle A1, ext(A1),$ {[0,5], [10,20], [40,*Now*]}$\rangle$, $\langle A2, ext(A2),$ {[10,20], [40,*Now*]}$\rangle$, ...}}

Analogously, for A^t_{inst}:
{[$\langle \alpha(L_{land},$ Land, parcelId, 'p1', [0,5] = P1$\rangle$, ..., $\langle \alpha(L_{land},$ Land, parcelId, 'p1', [10,20] = P1$\rangle$...]}

This expression indicates that, for instance, between instants 0 and 5, element 'p1' in level parcelId in dimension Land, is mapped to a spatial element 'P1'.

The fact tables are defined analogously to the non-temporal case.

3.2 Summary

In this chapter, the temporal data model was formally defined. We have presented the notion of *Lifespan*, *Temporal GIS-OLAP Dimension Schema* and *Temporal GIS-OLAP Dimension Instance* and we also provided an example that represents our proposal. These formal definitions constitutes the formal basics for a temporal query language, TPiet-QL, that we introduce next.

Chapter 4

Query language

We now present a query language based on the formal data model we have previously introduced in **Chapter 3**. Here we provide the syntax and semantics of the language and some examples that illustrate its use.

4.1 Temporal Piet Data Structure

The data structure of the temporal extension to the Piet data model, is organized in: (a) Application information. This is the data warehouse structure, and contains dimension and fact tables. (b) GIS information. Corresponds to the data structure for the map layers. Contains a table per layer. Temporal attributes FROM and TO indicate the interval of validity of each object in a layer. (c) GIS-OLAP mapping information. These are the data structures for storing the relationship between geometric and application information. (Actually these structures store the α functions). Temporal attributes are also included here. (d) There are also data structures to store precomputed information containing the overlay of different layers. We do not discuss this issue in this thesis. The interested reader is referred to [9]. The system supports the following changes, usually found in the literature [15] (a study of the implementation of these operators is outside the scope of this thesis): (a) Creation; (b) Update (e.g., a change in the object's extension); (c) Deletion (in the temporal sense, i.e., without losing the object's history); (d) Reincarnation (an object is deleted, and then re-created); (e) Split (an object generates two or more new ones); (e) Merge (two or more objects are merged into a new one). As we expressed above, the different states of spatial objects are materialized as usual in temporal databases, adding two attributes (denoted FROM and TO) of temporal type, to each one of the tables in the GIS-OLAP and GIS information parts. Let us briefly explain the semantics of the update operators. When a new object is *created* at instant t_1, say, in the layer *Land* (e.g., a new parcel is added), a tuple is inserted in the *Land* table, with the corresponding parcel information. The attributes FROM and TO are set to t1 and the distinguished value *Now*, respectively. Analogously, if this parcel, call it p_1; is *split* into p_2 and p_3 at instant t_2; the tuple for p_1 is *deleted* in the temporal database sense, that is, it is timestamped with

TO=$t_2 - 1$ (i.e., an instant immediately before t_2 in the object's granularity); in addition, two tuples are *created* for p_2 and p_3; with FROM=t_2; and TO=*Now*. Later, at t_4, two parcels, p_5 and p_6 are *merged* into a single one, call it p_{56}. The former two tuples are *deleted* as before (i.e., timestamped with TO=t_4-1), and p_{56} is *created* with FROM=t_4 and TO=*Now*. The *update* operation at instant t is equivalent to the *deletion* of a tuple (i.e., a timestamping with t-1), and the *insertion*, at instant t; of a new one (keeping the same identifier). The *reincarnation* operator is analogous to an update, except for the fact that the instants of deletion and insertion are not consecutive.

We now discuss the *data warehouse* side. One of the premises of our model is to allow autonomous maintenance of warehouse and GIS information, i.e., that OLAP data could be managed independently from the GIS information. There are at least two possible situations: (a) The data warehouse and associated data cubes are the usual ones, where fact tables are updated off-line (usually, only data insertion is allowed), and the dimensions are *static*, i.e., the history of the dimension members and hierarchies is not kept, and only the current state of the dimension data is available (i.e., the data warehouse is *non-temporal*). (b) A *temporal* warehouse is available, i.e., dimensions are updated and their history is preserved. One solution is to use the notion of *slowly changing dimensions*[13], where a new dimension tuple is added when an update occurs (dimension tables are extended with FROM/TO attributes). More sophisticated solutions can be found in the literature [7, 14]. Note that when the operations on the GIS side require creating new spatial objects (like in the *creation*, *split*, and *merge* operations), the corresponding objects must be inserted in the warehouse dimensions, also defining new mappings. However, when an *update* occurs (like a change in an object's shape) the object identifier does not change and no action needs to be taken on the warehouse side. Note that the mentioned insertions can be performed without impacting the warehouse or the mapping function, although this could produce incomplete answers to some queries (the ones that involve accessing the warehouse), due to the incomplete mapping (i.e., the object would only be in one of the parts of the system).

4.2 Temporal Functions and Predicates

Before introducing the query language, we discuss the functions it must support. We focus on the temporality of spatial objects, since the temporal databases literature abound in the treatment of temporal alphanumerical attributes.

Definition 4 *(Spatio-temporal object).* We denote by spatio-temporal object a tuple of the form $\langle objectId, geometry, attribute_1, \ldots, attribute_n, interval_i \rangle$ where geometry is the geometric extension of the object, $attribute_i$ are alphanumeric attributes, and 'interval' is the interval of validity of the object, of the form [FROM, TO].

Note that in Definition 4, *interval* is a single interval. In temporal databases it is usual to talk about temporal elements, i.e., sets of intervals. For simplicity of presentation, in this thesis we work with single intervals instead of temporal elements. Also for simplicity

we assume *valid time*, although bitemporal relations can be also supported. In what follows we refer to spatio-temporal objects as 'objects' when this can be inferred from the context, and denote $\mathcal{G}$ a collection of spatio-temporal objects. The basic constructs of the language are spatio-temporal objects. Therefore, based on Allen's interval set of predicates [1], we specify the syntax and semantics of a collection of predicates over spatio-temporal objects, intervals, and time instants.

- **StartsBefore(g,t)**: $\mathcal{G} \times T \rightarrow$ *boolean*: Given a spatio-temporal object and an instant, returns $True$ if $t >$ `g.FROM`.
- **BeginsAfter(g,t)**: $\mathcal{G} \times T \rightarrow$ *boolean*: Given a spatio-temporal object and an instant, returns $True$ if $t <$ `g.FROM`.
- **FinishesAfter(g,t)**: $\mathcal{G} \times T \rightarrow$ *boolean*: Given a spatio-temporal object and an instant, returns $True$ if $t <$ `g.TO`.
- **At(g,t)**: $\mathcal{G} \times T \rightarrow$ *boolean*: Given a spatio-temporal object and an instant, returns $True$ if $t \leq$ `g.FROM AND` $t \geq$ `g.TO`.
- **Before(g,**$\langle t_1, t_2 \rangle$**)**: $\mathcal{G} \times T \times T \rightarrow$ *boolean*: Given a spatio-temporal object and an interval, returns $True$ if `g.TO` $< t_1$.
- **After(g,**$\langle t_1, t_2 \rangle$**)**: $\mathcal{G} \times T \times T \rightarrow$ *boolean*: Given a spatio-temporal object and an interval, returns $True$ if $t_2 <$ `g.FROM`.
- **During(g,**$\langle t_1, t_2 \rangle$**)**: $\mathcal{G} \times T \times T \rightarrow$ *boolean*: Given a spatio-temporal object and an interval, returns $True$ if $t_1 \leq$ `g.FROM AND` $t_2 \geq$ `g.TO`.
- **Overlaps(g,**$\langle t_1, t_2 \rangle$**)**: $\mathcal{G} \times T \times T \rightarrow$ *boolean*: Given a spatio-temporal object and an interval, returns $True$ if ($t_1 <$ `g.FROM AND` $t_2 >$ `g.FROM AND` $t_2 <$ `g.TO`) OR ($t_1 >$ `g.FROM AND` $t_2 >$ `g.TO AND` $t_1 <$ `g.TO`)
- **Covers(g,**$\langle t_1, t_2 \rangle$**)**: $\mathcal{G} \times T \times T \rightarrow$ *boolean*: Given a spatio-temporal object and an interval, returns $True$ if $t_1 \geq$ `g.FROM AND` $t_2 \leq$ `g.TO`.
- **Meets(g,**$\langle t_1, t_2 \rangle$**)**: $\mathcal{G} \times T \times T \rightarrow$ *boolean*: Given a spatio-temporal object and an interval, returns $True$ if $t_1 =$ `g.TO OR` $t_2 =$ `g.FROM`.
- **Starts(g,**$\langle t_1, t_2 \rangle$**)**: $\mathcal{G} \times T \times T \rightarrow$ *boolean*: Given a spatio-temporal object and an interval, returns $True$ if ($t_1 =$ `g.FROM AND` $t_2 >$ `g.TO`) OR ($t_1 =$ `g.FROM AND` $t_2 <$ `g.TO`)
- **Finishes(g,**$\langle t_1, t_2 \rangle$**)**: $\mathcal{G} \times T \times T \rightarrow$ *boolean*: Given a spatio-temporal object and an interval, returns $True$ if ($t_1 <$ `g.FROM AND` $t_2 =$ `g.TO`) OR ($t_1 >$ `g.FROM AND` $t_2 =$ `g.TO`)

Note that `DURING` and `COVERS` represent the predicates `X DURING Y` in Allen's algebra. `OVERLAPS` represents `X OVERLAPS Y` and `Y OVERLAPS X`. The same applies to `MEETS`, `STARTS` and `FINISHES`. `BEFORE` and `AFTER` represent `X < Y` and `Y < X`, respectively. We also need the **Coalesce** function defined next.

Coalesce($\mathcal{G}$): The function returns a collection of spatio-temporal objects, where these objects are obtained by merging into a single one all objects whose temporal intervals are consecutive and that coincide in all other attributes. This is analogous to the well-known coalesce function in temporal databases.

4.3 Spatio-temporal Joins

A key operation in any spatio-temporal query language is the join. Different kinds of temporal joins have been proposed in the literature [18], and two main classes can be identified: (a) Disjoint join; and (b) Overlap join. In the former, given n (timestamped) tuples, it is not required that their time intervals overlap. In the latter, the time intervals must overlap and there are two possibilities: all the time intervals have at least one common time instant, or they are joined in a chained fashion, e.g. $t_1.TO \geq t_2.FROM \wedge t_2.TO \geq t_1.TO$. Disjoint joins provide more expressiveness to a query language than overlap joins, allowing to query for asynchronous events (e.g., parcels owned by X before a region changed name). Examples (following Allen [1]) are `before-join(X,Y)`, and `meet-join(X,Y)`, with conditions $X.TO \geq Y.FROM$ and $X.TO = Y.FROM$, respectively. Analogously, the condition for the (symmetric) overlap join is $X.FROM < Y.TO \wedge Y.FROM < X.TO$.

The joins above are denoted T-joins. When a T-join requires the equality of a collection of non-temporal attributes specified as a predicate P_a, we say that we are in presence of a GT-join (standing for generic temporal). That is, a GT-join corresponds to the expression $\sigma_{P_a \wedge overlap-join(X,Y)}$(X,Y). That means, given two tuples, the tuples in the result of a GT-join will be the ones that have overlapping time intervals and verify the non-temporal predicate P_a. In a spatio-tempoal setting we can implement the temporal joins using the operators defined above. For example, if we have a layer denoted Parcels, a before-join between two objects can be defined as:

```
SELECT p1
FROM Parcels p1, parcels p2
WHERE BEFORE (p1,p2.FROM) AND
p1.owner='Peter' AND p2.owner ='John' and p1.id=p2.id
```

In this query we are asking for the parcels that Peter owned before John did. Below we give more details on the language. Analogously, a meet join would be obtained by replacing the predicate `BEFORE(p1,p2.FROM)` with `MEETS(p1,p2.FROM)`.

In the presence of spatio-temporal objects, the GT-join can be defined using the standard topological relationships in [8], like `Touches`(g_1, g_2) or `Contains`(g_1, g_2).

Consider two layers storing the histories of airports and cities:

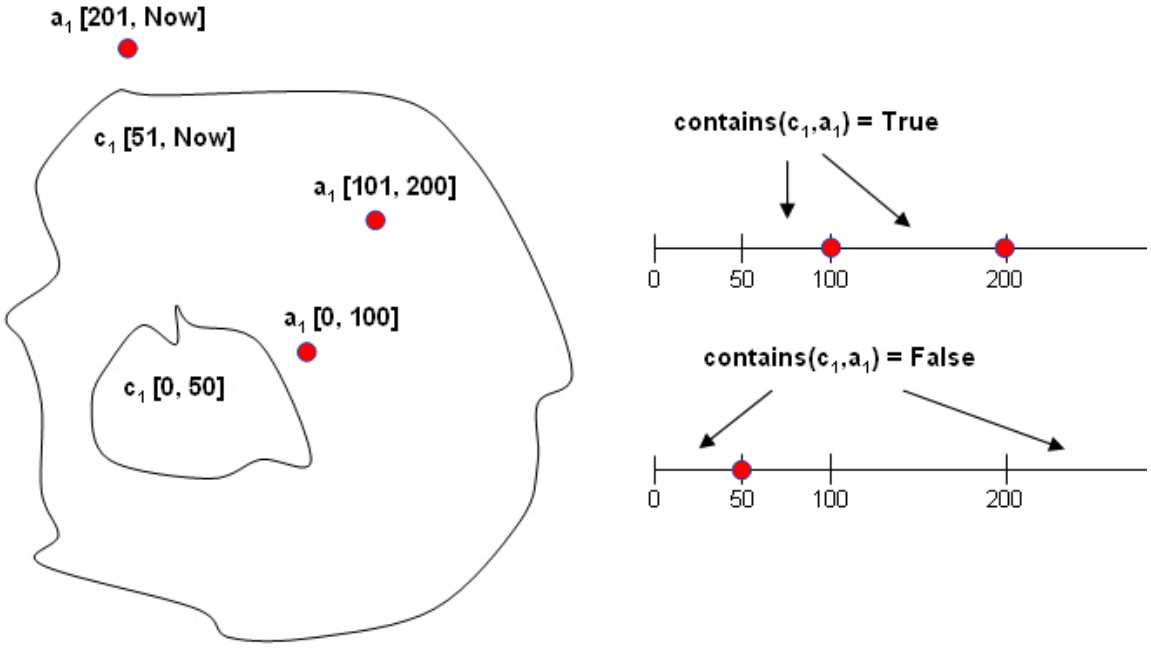


Figure 4.1: A city and its airport (left); Interactions of a_1 and c_1 along their timelines (right)

Figure 4.1 (left) shows two stages of city c1: one in the interval [0,50], and the other in the interval [51,Now]. Airport a1 was first relocated at instant 100, and then, due to the city expansion, it was relocated outside the new city limits. Figure 4.1 (right) shows how the two objects a1 and c1 interact along their timelines: the airport is within the city limits only in the intervals [51,100] and [101,200].

The relational representations are given below:

cityId	the_geom	...	FROM	TO
c1	g1	...	0	50
c1	g2	...	51	Now
c2	g3	...	0	30

airportId	the_geom	...	FROM	TO
a1	g1	...	0	50
a1	g2	...	51	Now
...	...	...	...	...

We can list the pairs *city-airport* such that an airport was within the city limits as an GT-join, where the non-temporal predicate *Contains* is spatial:

$\sigma_\phi(Airports \times Cities)$
$\phi = contains(Airports.geom, Cities.geom) \wedge overlap - join(Airports, Cities)$

The result would contain the tuples $\langle a1, c1, 51, 100\rangle$ and $\langle a1, c1, 101, 200\rangle$, representing that between instants 51 and 100, a1 remained within the city limits of a1 (see Figure 4.1).

We could write a more concise version of this query using a spatio-temporal semantics for the predicate *Contains*, where the predicate is evaluated only in the overlapping time intervals. In practice, we believe that using the non-temporal semantics for the spatial predicates is more flexible and does not require new definitions in a spatial GIS. Therefore, we use this semantics in TPiet-QL, as we explain below.

4.4 The TPiet-QL Query Language

The discussion above set the basis for defining a temporal extension to Piet-QL.

```
SELECT GIS [[DISTINCT] SNAPSHOT]] list of attributes and/or functions
FROM [OVERLAP] T1 t1,...,Tn tn
WHERE Φ
```

Let us explain this syntax. The list of attributes in the `SELECT` clause (if necessary distinguished by prefixing it by the name or alias of the corresponding table) and any of the temporal or geometric functions specified in Tables 4.1 and 4.3 can be used. If any of these functions are selected, an *alias* should be added to the query (*this is mandatory*). Each tuple in the result set has an implicit valid interval, that corresponds to the *CommonInterval* function. Furthermore, the result set is automatically *coalesced*.

The optional `SNAPSHOT` clause can be used to project the *CommonInterval* over the interval (-∞, ∞). Internally, a coalesce is applied previous to returning the valid interval. Thus, duplicate tuples can be obtained in the result set. The `DISTINCT` reserved word is an option to avoid obtaining this dupplication. The `GIS` reserved word is used by compatibility with Piet-QL [10].

`T1` through `Tn` represent thematic layers, `t1` through `tn` range over the spatial or *spatiotemporal objects* in these layers, and the a_i's represent attributes of these objects. Different kinds of temporal joins have been proposed in the literature [18], and two main classes can be identified:

- Overlap join: It is expressed by the clause `FROM OVERLAP`. Time intervals of the objects must overlap. The language also allows the use of static objects in the join. In this case, we consider that static objects have implicit validity interval (-∞, ∞). The

use of this clause is similar to adding, in the `WHERE` clause, several *Overlaps* temporal predicates among all the temporal objects used in this join. Nevertheless, the use of Overlap Join is not equivalent to using several Overlaps temporal predicates in the `WHERE` clause, when using with the *CommonInterval(), CMStart() and CMEnd()* temporal functions as explained in Table 1.

- Disjoint join: It is expressed by the clause `FROM`. This provides more expressiveness to the query language than the Overlap Join, allowing to query for asynchronous events. Typically, temporal predicates can be added in the `WHERE` clause to compare temporal intervals.

Note that only table names can be used. No nested queries can be used in the `FROM` clause.

`CommonInterval()`	If *Overlap Join* is used, this function returns the intersection of all valid object's intervals that participate in the `FROM OVERLAP` clause, considering that static objects have infinity validity. When *Disjoint join* is selected, it returns the valid interval of the first spatio-temporal objects that appear in the `FROM` clause, from left to right. If no spatio-temporal objects appear, all the objects are static and ($-\infty$, ∞) interval is returned.
`CMStart()`	Returns the first component of CommonInterval() function.
`CMEnd()`	Returns the last component of CommonInterval() function.
`Now()`	Returns the literal convention for representing ∞. It is useful for checking if a spatio-temporal objects is still in force.
`current_date`	Returns the current *sysdate* of the system. It is invoked without parenthesis, like a variable.

Table 4.1: Temporal Functions over spatio-temporal joins

`Within(A,B)`, `ST_Within(A,B)`	Returns *True* if the geometry A is completely inside geometry B.
`Contains(A,B)`, `ST_Contains(A,B)`	Returns *True* if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A.
`Intersects(A,B)`, `ST_Intersects(A,B)`	Returns *True* if the Geometries "spatially intersect" - (share any portion of space) and *False* if they don't (they are *Disjoint*).
`Crosses(A,B)`, `ST_Crosses(A,B)`	Returns *True* if the supplied geometries have some, but not all, interior points in common.
`Touches(A,B)`, `ST_Touches(A,B)`	Returns *True* if the geometries have at least one point in common, but their interiors do not intersect.
`Disjoint(A,B)`, `ST_Disjoint(A,B)`	Returns *True* if the Geometries do not "spatially intersect" - if they do not share any space together.
`Equals(A,B)`, `ST_Equals(A,B)`	Returns *True* if the given geometries represent the same geometry. Directionality is ignored.
`CoveredBy(A,B)`, `ST_CoveredBy(A,B)`	Returns *True* if no point in Geometry A is outside Geometry B.
`Overlaps(A,B)`, `ST_Overlaps(A,B)`	Returns *True* if the Geometries share space, are of the same dimension, but are not completely contained by each other.
`Covers(A,B)`, `ST_Covers(A,B)`	Returns *True* if no point in Geometry B is outside Geometry A.
`OrderingEquals(A,B)`, `ST_OrderingEquals(A,B)`	Returns *True* the given geometries represent the same geometry and points are in the same directional order.
`IsClosed(A)`, `ST_IsClosed(A)`	Returns *True* if the LINESTRING's start and end points are coincident. For Polyhedral surface is closed (volumetric).
`IsEmpty(A)`, `ST_IsEmpty(A)`	Returns *True* if this Geometry is an empty geometry.

Table 4.2: Geometric Predicates over geometries

`IsSimple(A)`, `ST_IsSimple(A)`	Returns *True* if this Geometry has no anomalous geometric points, such as self intersection or self tangency.
`IsValid(A)`, `ST_IsValid(A)`	Returns *True* if the Geometry is well formed.

Table 4.2 (cont): Geometric Predicates over geometries.

The condition Φ is composed of conjunctions and disjunctions of functions and predicates mentioned above. In this clause we add boolean expressions that can be built by using nested `AND`, `OR` and `NOT` operators. The predicates can be:

- A subset of the Simple Features for SQL specification from the Open Geospatial Consortium[1] for geometric objects. For more details see Table 4.2 and Table 4.3.
- Temporal functions for recovering overlapped temporal intervals, shown in Table 4.1.
- Temporal predicates based on Allen's interval set of predicates [1] for comparing spatio-temporal objects, intervals and time instants. For more details see Table 4.3.
- Comparison predicates ($=$, $<$, $\leq$, $>$, $\geq$ and $<>$).
- A subset of the `IN` predicate [10] where it is allowed to nest several *GIS* queries and only the more nested expression can be an *OLAP* query (e.g.: this query cannot embed any other nested query). This restriction differs from the original Piet-QL where any kind of nested combinations were allowed. We simplified this in TPiet-QL since we put emphasis on temporal *GIS* queries by adding new temporal predicates.

`Area(A)`, `ST_Area(A)`	Returns the area of the surface if it is a polygon or multi-polygon.
`Envelope(A)`, `ST_Envelope(A)`	Returns Returns a geometry representing the double precision (float8) bounding box of the supplied geometry.
`Relate(A,B)`, `ST_Relate(A,B)`	Given two geometries A and B, returns a text that represents intersections between the Interior, Boundary and Exterior of A and B, according to the `DE-9IM` matrix.

Table 4.3: Geometric Functions over geometries.

[1] http://www.opengeospatial.org

`Distance(A,B)`, `ST_Distance(A,B)`	Returns the 2-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units.
`Intersection(A,B)`, `ST_Intersection(A,B)`	Returns a geometry that represents the shared portion of geom A and geom B.
`Difference(A,B)`, `ST_Difference(A,B)`	Returns a geometry that represents that part of geometry A that does not intersect with geometry B.
`Union(A,B)`, `ST_Union(A,B)`	Returns a geometry that represents the point set union of the Geometries.
`Length(A)`, `ST_Length(A)`	Returns the 2d length of the geometry if it is a linestring or multilinestring.
`Dimension(A)`, `ST_Dimension(A)`	Returns the inherent dimension of this Geometry object, which must be less than or equal to the coordinate dimension.
`Simplify(A)`, `ST_Simplify(A)`	Returns a "simplified" version of the given geometry using the Douglas-Peucker algorithm.
`Boundary(A)`, `ST_Boundary(A)`	Returns the closure of the combinatorial boundary of this Geometry.
`Box(A)`, `ST_Box(A)`	Returns a Polygon geometry representation of the 2D or 3D bounding box.
`Centroid(A)`, `ST_Centroid(A)`	Returns the geometric center of a geometry.
`Perimeter(A)`, `ST_Perimeter(A)`	Return the length measurement of the boundary of an ST_Surface or ST_MultiSurface geometry.
`X(A)`, `ST_X(A)`	Return the X coordinate of the point, or NULL if not available. Input must be a point.
`Y(A)`, `ST_Y(A)`	Return the X coordinate of the point, or NULL if not available. Input must be a point.

Table 4.3 (cont): Geometric Functions over geometries.

The semantics of the query is defined by the cartesian product of the geometric objects in all the thematic layers in the `FROM` clause. If the `OVERLAP` keyword is specified, only the tuples whose intervals overlap are considered, (ie., the tuples such that $\cap_{ti.interval,i=1,n} \neq \emptyset$), and the overlapping interval are included in the result, which is *coalesced* by default using all the non-temporal attributes in the `SELECT` clause. The *coalesce* operation is defined as follows. Given a collection of objects ($\mathcal{G}$), for all objects that match their non-temporal

attributes and whose temporal intervals are consecutive, **Coalesce($\mathcal{G}$)** constructs a single spatio-temporal object composed by non-temporal attributes and the temporal union of all the intervals.

We illustrate the semantics extending the city-airport example with a layer containing parcels, described in the table below (on the right we show the distances between cities and parcels during different time intervals, although this information is actually not recorded, it must be computed):

parcelId	the_geom	...	FROM	TO
p1	g1	...	10	20
p1	g2	...	21	40
p2	g3	...	30	50
p3	g4	...	40	100
...	...	...	...	...

cityId	parcelId	FROM	TO	distance
c1	p1	10	20	80
c1	p1	21	40	120
c1	p2	30	50	70
c1	p3	40	50	80
c1	p3	51	100	90

Consider a TPiet-QL query asking for pairs city-parcel such that the distance between them is/was less than 100 Km. According to the usual semantics of a temporal join, this query returns tuples of the form $\langle p_i, c_j, Interval \rangle$, where *Interval* is the temporal interval when they where closer than 100 Km from each other.
The query reads in TPiet-QL:

```
SELECT GIS c, p
FROM OVERLAP Parcels p, Cities c
WHERE Distance(c.the geom, p.the geom) < 100
```

The result will be (note that this result is coalesced):

cityId	parcelId	FROM	TO
c1	p1	10	20
c1	p2	30	50
c1	p3	40	100

The next example includes an OLAP subquery in the `WHERE` clause (technically, in TPiet-QL this is called a GIS-OLAP query): We assume the existence of an external data cube denoted Weather, with dimensions *Geography* and *Time*, and measure precipitation, representing the precipitation per year.

Query 1: *Protected areas with a surface larger than 100 Ha in 1996, currently larger than at that time, with a precipitation higher than 120 mm in 2010.*

```
SELECT GIS p1.id
    FROM pAreas p, pAreas p1
    WHERE area(p) > area(p1) AND
        COVERS(p,[1996,1996]) AND
        COVERS(p1,[2012,2012]) AND
        p1.id = p.id AND
        p1.id IN(SELECT
                CUBE filter([Geography].[Geography areaId].Members,
                [Measures].[precipitation] > 120)
                FROM [Weather]
                SLICE [Time].[2010])
```

We can see the constructs of the formal language in the TPiet-QL expression above. The main difference is that instead of using non-temporal functions over the extensions of spatial objects, while *area* is applied over a geometry (e.g., e_p), TPiet-QL uses temporal functions over spatio-temporal objects (e.g., p).

4.5 TPiet-QL Syntax

We conclude this chapter presenting the complete TPiet-QL syntax. This syntax is formally depicted in Listing 1.

This syntax uses the following atoms:

- `FUNCTION_STPQL$`: any function in Table 4.1 and Table 4.3 or boolean predicate in Table 4.2 and Table 4.3. Functions cannot be nested.
- `LAYER$`: references a layer in the GIS.
- `LAYER_ALIAS$`: literal.
- `NUMBER$`: any non-negative Integer.
- `GENERAL_PREDICATE$`: any General predicate ($=$, $<$, $\leq$, $>$, $\geq$). Operands can be functions.
- `GIS_PREDICATE$`: any predicate in Table 4.2 and Table 4.3.
- `GIS_ATTRIBUTE$`: references an attribute associated to a layer.
- `OLAP_CUBE$`: references an OLAP data cube.

- OLAP_MEMBER$: references a member in an OLAP data cube.
- OLAP_LEVEL$: references a level in an OLAP data cube.
- FUNCTION_O$: the OLAP function Filter and function with no arguments like Children and Members.

Listing 1: TPiet-QL Syntax

```
query_ = SELECT GIS [[DISTINCT] SNAPSHOT] gislist_ fromclause_g_
[where_clause_g_ ] [LIMIT NUMBER$]
gislist_ = attribute_path_g_ [AS attribute_g_ ] [, gislist_ ]
gislist_ = FUNCTION_STPQL$ [AS attribute_g_ ] [, gislist_ ]
gislist_ = [LAYER$.]* [, gislist_ ]
gislist_ = [LAYER_ALIAS$.]* [, gislist_ ]
attribute_path_g_ = [LAYER$.] attribute_g_
attribute_g_ = GIS_ATTRIBUTE$
fromclause_g_ = FROM [OVERLAP] tablelist_
tablelist_ = LAYER$ [AS LAYER_ALIAS$] [, tablelist_ ]
where_clause_g_ = WHERE gis_filter_
gis_filter_ = gis_predicate_ [AND gis_filter_ ]
gis_filter_ = gis_predicate_ [OR gis_filter_ ]
gis_filter_ = NOT(gis_filter_)
gis_filter_ = (gis_filter_)
gis_predicate_ = GENERAL_PREDICATE$
gis_predicate_ = GIS_PREDICATE$
gis_predicate_ = attribute_path_g_ IN (single_result_subquery_)
single_result_subquery_ = SELECT subq_
subq_ = GIS [[DISTINCT] SNAPSHOT] gislist_ fromclause_g_
[where_clause_g_ ] [LIMIT NUMBER$]
subq_ = CUBE olaplist_ from_clause_o [sliceclause_ ]
olaplist_ = attribute_path_o [, olaplist_ ]
attribute_path_o = attribute_o.attribute_path_o
attribute_o = OLAP_MEMBER$ | OLAP_LEVEL$ | FUNCTION_O$
from_clause_o = FROM OLAP_CUBE$
sliceclause_ = SLICE slicefilterlist_
slicefilterlist_ = attribute_path_ [, slicefilterlist_ ]
```

4.6 Summary

We have presented a query language, denoted TPiet-QL, which consists of a temporal extension to the Piet data model and can express spatio-temporal integrated GIS-OLAP queries in a natural an concise way. TPiet-QL supports the operators proposed by the Open Geospatial Consortium for SQL. In addition, the syntax incorporates temporal functions and predicates to integrate with OLAP operations through the OLAP standard MDX. We gave detail of TPiet-QL syntax and semantics and provided some queries as example. In **Chapter 7**, we will provide an extensive set of queries that shows the power of the language and its main features.

Chapter 5

Implementation

For the implementation of the present thesis, based on the theory described above, a TPiet-QL API has been designed and implemented. It can be embedded in the Java host language and once a TPiet-QL query is executed, XML metadata and a temporary table are generated, containing the result set. In this chapter we describe the implementation details of this TPiet-QL API and the query engine.

5.1 TPiet-QL Engine and TPiet-QL Programmatic API

TPiet-QL engine was built using different software technologies which interact within each other to accomplish the query parsing and execution at the back-end.

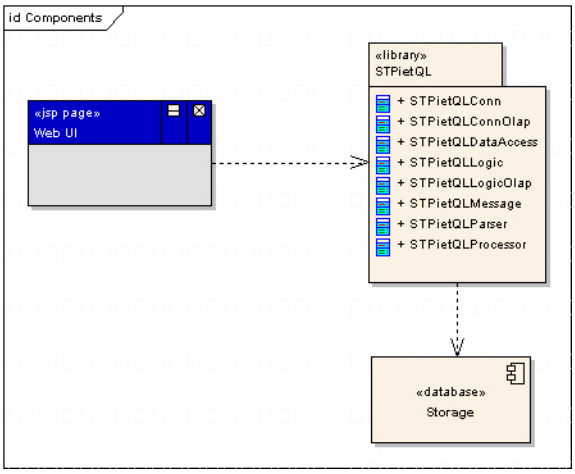


Figure 5.1: TPiet-QL: Components Diagram

The engine was developed by using the Java programming language V1.6[1]. Data are stored in a PostgreSQL V9.0[2] database with the Postgis V1.5[3] plug-in, which enables spatial functionalities. Besides, the OLAP Server is Mondrian V3.2[4]. The core Java classes and their functionalities of TPiet-QL engine are depicted in Figure 5.2.

The overall architecture of this back-end and the components interaction are shown through Sequence Diagram in Figure 5.3.

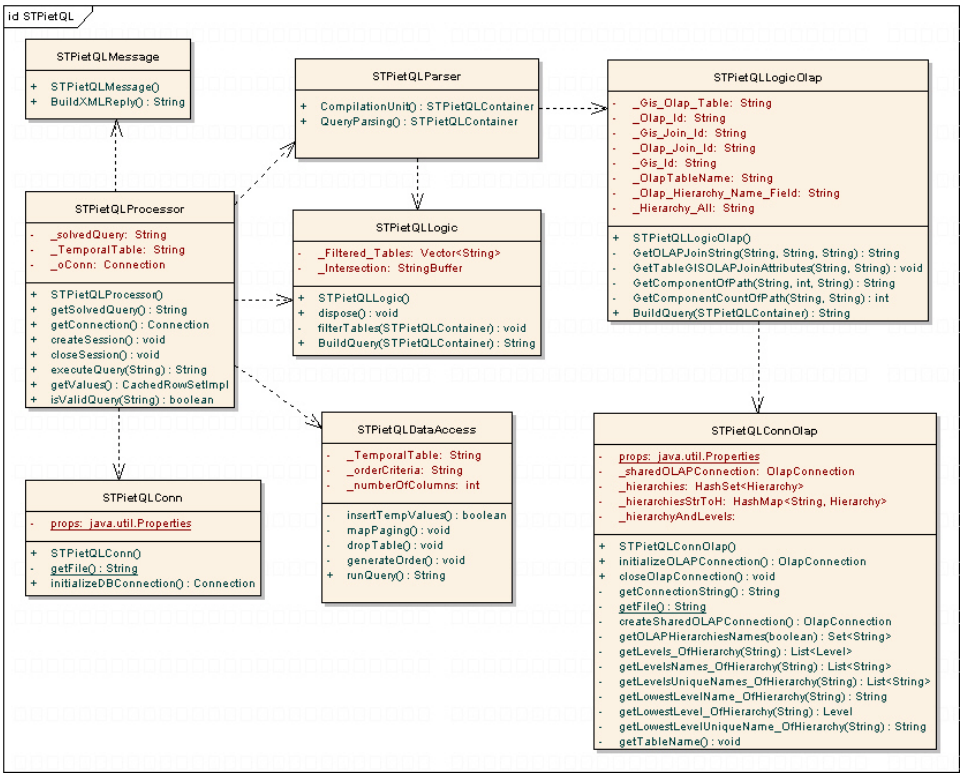


Figure 5.2: TPiet-QL: Classes Diagram

[1]http://www.oracle.com/technetwork/java/javase/overview/index-jsp-136246.html/
[2]http://www.postgresql.org/
[3]http://postgis.refractions.net/
[4]http://mondrian.pentaho.com/

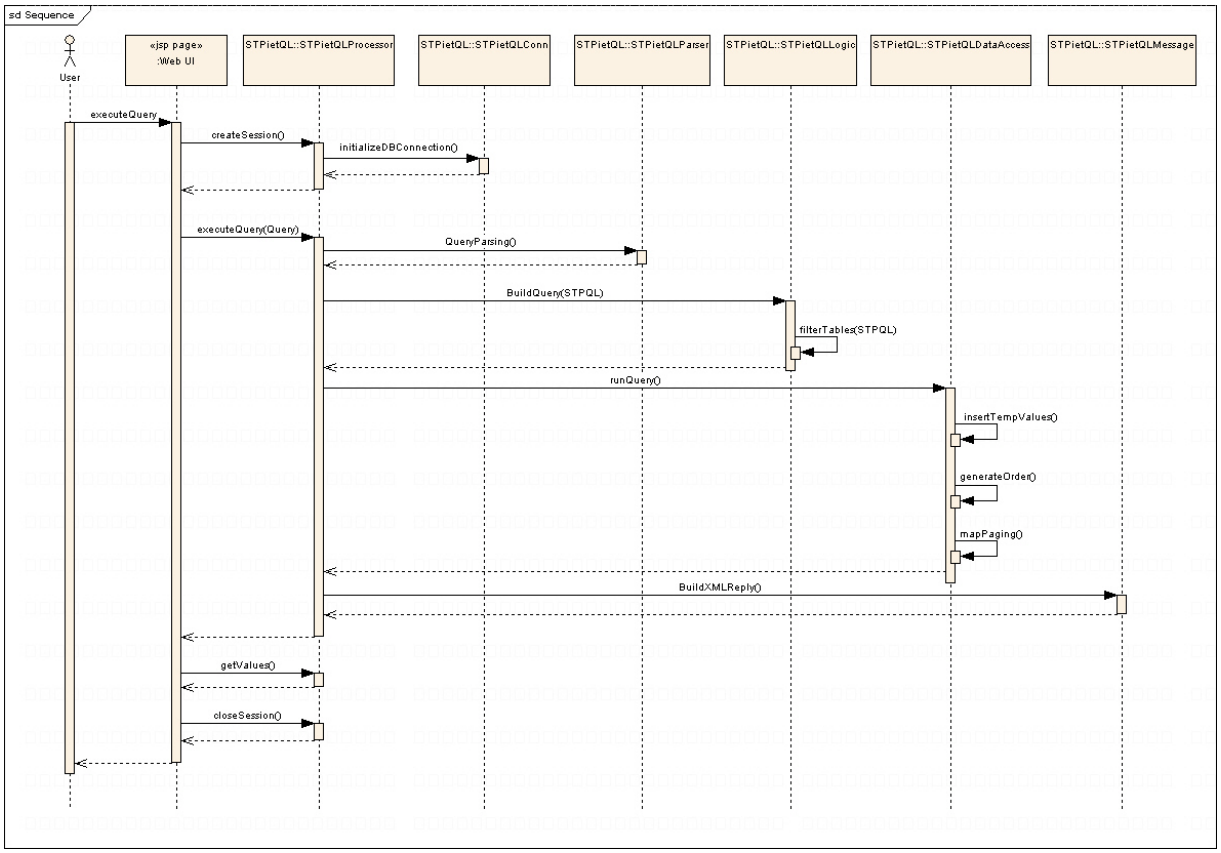


Figure 5.3: TPiet-QL: Sequence Diagram

5.2 TPiet-QL Query execution

Each time a TPiet-QL query is executed, the engine proceeds in two sequential steps. First of all, it *parses* the query according to the syntax defined in Section 4.5 and identifies those query parts which correspond to GIS and those which correspond to SOLAP with the goal of transforming and preparing them for execution. More precisely, those query *sections* that start with the `SELECT GIS` clause are translated into SQL with Postgis extension and those that start with the `SELECT CUBE` clause are translated into MDX dialect. If during this phase a parsing error is detected, a Java exception is returned to the invoker. Otherwise, the second phase takes place. That is, TPiet-QL engine executes SQL parts in the PostgreSQL database and the MDX expressions in the Mondrian OLAP engine. Finally, it combines intermediate result sets and coalesces them.

The result set is composed of two parts: (a) an *XML document* that describes the metadata; (b) a *temporary table* which contains the tuples with the resulting data. Typically, a client application connects to the engine in order to execute a TPiet-QL query. The engine, according to what we have mentioned above, returns a Java exception or an

XML document containing the metadata. If the execution was successful, it also generates a temporary table. Both, its name and its dynamic structure (i.e. which depends on the query) are embedded in this XML document. The client application parses the XML document because it contains all the information necessary about the execution.

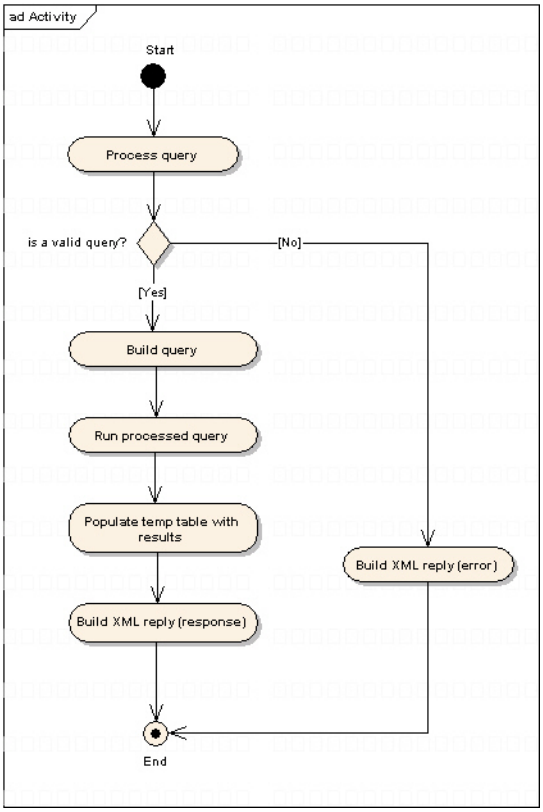


Figure 5.4: TPiet-QL: Activity Diagram

Finally, this table can be queried to achieve the proposed goal, e.g. show the result set over a map. The following XML schema, shown in Listing 2, describes the metadata returned by the engine to a client application:

Listing 2: XML Schema for Resultset

```
<?xmlversion='1.0'encoding='UTF-8'?>
<xs:schemaxmlns:xs='http://www.w3.org/2001/XMLSchema'
elementFormDefault='qualified'>
 <xs:element name='query'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='status'/>
    <xs:element ref='additionalinfo'/>
   </xs:sequence>
  </xs:complexType>
  </xs:element>
  <xs:element name='status'>
  <xs:complexType>
   <xs:sequence>
    <xs:element name='code' type='xs:long'/>
    <xs:element name='severity' type='xs:string'/>
    <xs:element name='message' type='xs:string'/>
    <xs:elementname='date' type='xs:string'/>
   </xs:sequence>
  </xs:complexType>
  </xs:element>
  <xs:element name='additionalinfo'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='performance'/>
    <xs:element ref='results'/>
   </xs:sequence>
  </xs:complexType>
  </xs:element>
  <xs:element name='performance'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='parse_elapsed'/>
    <xs:element ref='query_elapsed'/>
   </xs:sequence>
  </xs:complexType>
  </xs:element>
```

```
<xs:element name='parse_elapsed'>
<xs:complexType>
 <xs:simpleContent>
 <xs:extension base='xs:string'>
  <xs:attribute name='unit' use='required' type='xs:string'/>
 </xs:extension>
 </xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name='query_elapsed'>
<xs:complexType>
 <xs:simpleContent>
 <xs:extension base='xs:string'>
  <xs:attribute name='unit' use='required' type='xs:string'/>
 </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:element name='results'>
<xs:complexType>
 <xs:sequence>
  <xs:element ref='description'/>
  <xs:element ref='table'/>
 </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name='description' type='xs:string'/>
<xs:element name='table'>
<xs:complexType>
 <xs:sequence minOccurs='0'>
<xs:element name='name' type='xs:string'/>
<xs:element name='rows' type='xs:long'/>
 <xs:element ref='columns'/>
 </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name='columns'>
<xs:complexType>
 <xs:sequence>
  <xs:element maxOccurs='unbounded' ref='column'/>
```

```
   </xs:sequence>
  </xs:complexType>
  </xs:element>
  <xs:element name='column'>
  <xs:complexType>
   <xs:attribute name='alias' use='optional' type='xs:string'/>
   <xs:attribute name='original_name' use='optional'
type='xs:string'/>
   <xs:attribute name='temp_name' use='optional' type='xs:string'/>
   <xs:attribute name='type' use='optional' type='xs:optional'/>
  </xs:complexType>
  </xs:element>
  </xs:schema>
```

5.3 Summary

In this chapter we have described the architecture of the solution starting from a conceptual vision, and then describing the specific architecture finally implemented. Also included is a description of the software components that are part of query engine implementation.

Chapter 6

Case Study

In the following chapter, we describe all the datasets we have prepared and used as scenario for our case study, based on real data with three types of information: Static GIS (that does not change over time), Temporal GIS (which evolves over time) and SOLAP data cubes.

6.1 Protected Areas

The International Union for Conservation of Nature (IUCN)[1] defines a protected area as "...is a clearly defined geographical space, recognized, dedicated and managed, through legal or other effective means, to achieve the long term conservation of nature with associated ecosystem services and cultural values."[2]. The World Database on Protected Areas (WDPA)[3] is a joint venture between the United Nations Environment Programme and the IUCN, and constitutes the largest assembly of data on the world's terrestrial and marine protected areas (PAs), containing 147.897 protected areas as of July 2010, with records for every country in the world. Those efforts needs sophisticated strategies and tools for effectively managing and monitoring of PAs, contributing to decision making support.

In particular, in Uruguay, the national law #17234 creates the National Protected Area System (NPAS) and defines it as the collection of natural areas of any kind (i.e., sea, land, etc.) in the nation's territory, that, due to their environmental, historical, cultural, or natural values, deserve to be preserved as national heritage, even if they have been modified by human actions. Clearly, the protected areas will evolve in time, as new areas will be added to the system and management plans are implemented, at the national and global level. Also, new information will be added at any time, and this should be supported by an appropriate information system. In addition, the status of the protected areas must be monitored (for instance, water quality, species diversity), and we would like to be able to analyze this temporal information in a user-friendly web-enabled interface.

[1] http://www.iucn.org/

[2] http://www.iucn.org/about/work/programmes/pa/pa_what/

[3] http://www.wdpa.org/

6.2 Datasets

As we have seen in previous chapters, TPiet-QL is a query language that allows querying in a same expression: Static GIS, Temporal GIS and SOLAP data cubes. We next describe all the datasets we have selected to illustrate the use of TPiet-QL.

6.2.1 Static GIS

The Normalized Difference Vegetation Index (NDVI) aims at measuring the live green vegetation in areas remotely sensed. Baeza *et al.*[3] classify ecosystems of Uruguay on the basis of three attributes derived from the seasonal curve of the NDVI using two decades of images of satellites. According to this work, Uruguay contains six Functional Ecosystem Types (TFEs), i.e. Ac2, Bf4, Ce11, Db11, Ed4, Fa2 and anomalous values (V Anom). They can be visualized in Figure 6.1. Table 6.1 shows nominal and numeric values of these TFEs, stored in tipo_func and grid_code attributes, respectively.

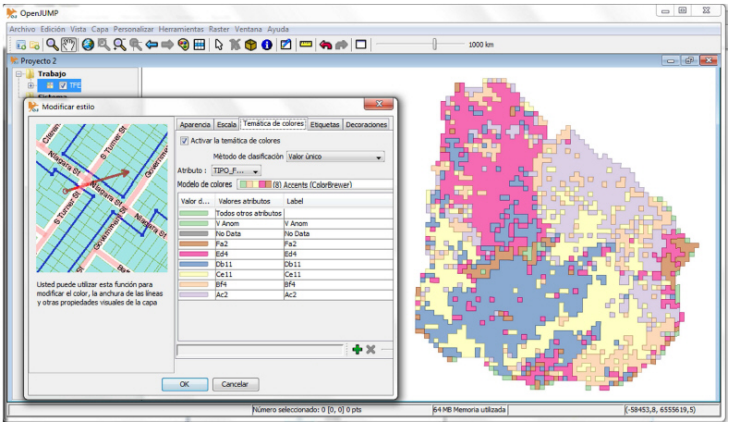


Figure 6.1: Functional Ecosystem Types Static Map

tipo_func	grid_code
Ac2	33
Bf4	36
Ce11	23
Db11	4
Ed4	30
Fa2	10
V Anom	3
No Data	0

Table 6.1: Functional Ecosystem Types nominal and numeric values

Arballo and Cravino[2] classify Uruguay into ten Ornito ecological zones, whose names are shown in Table 6.2. Figure 6.2 displays the Uruguay map divided into geo-referenced

spatial objects, each one belonging to a unique Ornito ecological zone. Both maps were published in geographical coordinated projection (spherical longitude/latitude) and downloaded from the Dirección Nacional de Medio Ambiente de Uruguay Website[4]. In order to analyze this information in a unified spatial view we carried out the transformation to the Universal Transverse Mercator System (UTM x/y coordinates).

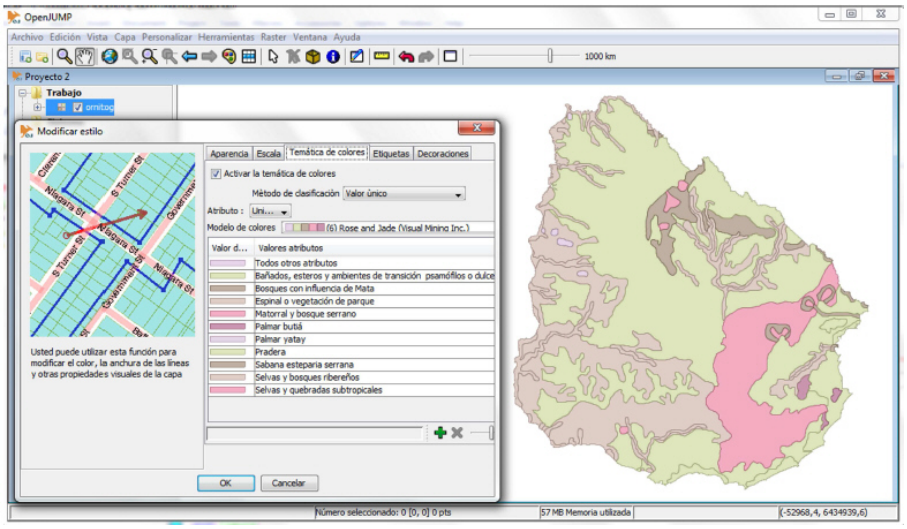


Figure 6.2: Ornito ecological Static Map of Uruguay

Name
Bañados
Bosques con influencia de Mata
Esteros y ambientes de transición psamófilos o dulceacuícolas
Matorral y bosque serrano Espinal o vegetación de parque
Palmar butiá
Palmar yatay
Pradera
Sabana esteparia serrana
Selvas y bosques ribereños
Selvas y quebradas subtropicales

Table 6.2: Names of the ornito geo zones in Uruguay.

Finally, for our examples, we use a map that contains the administrative division of Uruguay into 19 departments whose IDs and names are shown in Table 6.3. Figure 6.3 depicts an Uruguayan map, geometrically divided into its departments.

From now on, we consider these three static layers for our queries, named TFE, ornitogeo, and departament, respectively.

[4]http://www.dinama.gub.uy/geonetwork/srv/es/main.home

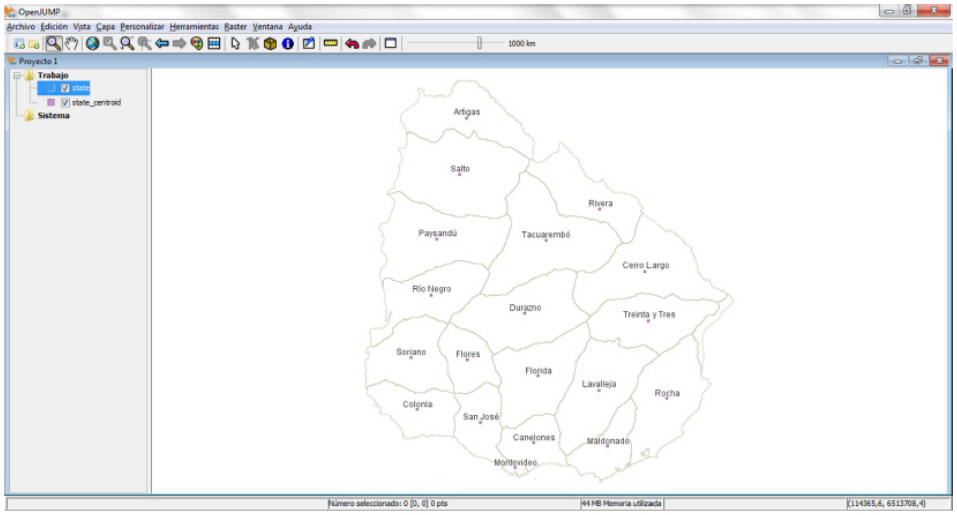


Figure 6.3: Administrative Division Static Map

- TFE table contains 508 tuples. The attributes we will use are 'tipo_func' and 'geom'. Its geometry is Polygon type.
- Ornitogeo table contains 108 tuples. The attributes we will use are 'unidad' (the name of an Ornito ecological zone) and 'geom'. Its geometry is Multi-polygon type.
- Department contains 19 tuples. The attributes we will use are 'name' and 'geom'. Its geometry is Multi-polygon type, since Uruguayan geography contains islands.

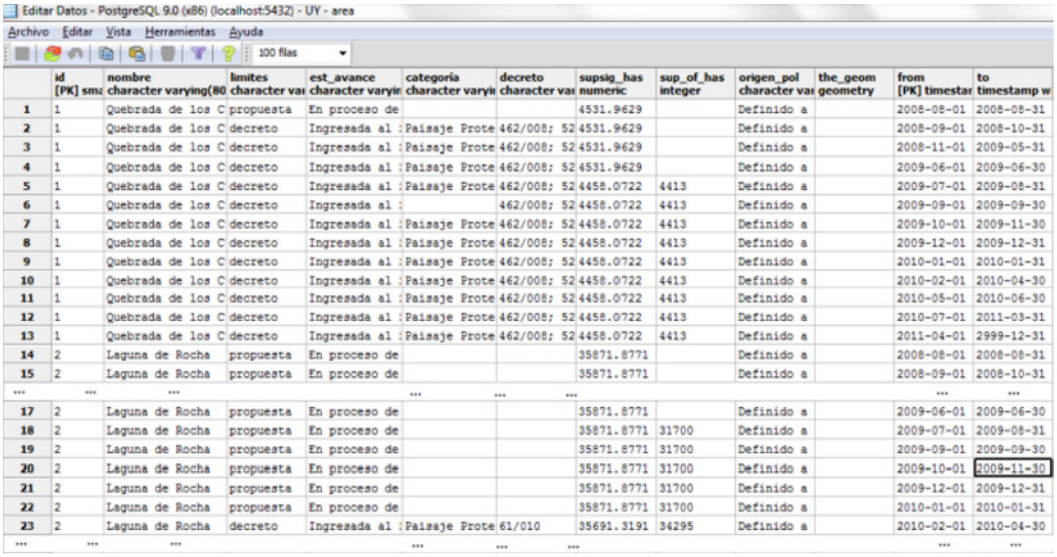
6.2.2 Temporal GIS

The real data we used was provided by the 'Sistema Nacional de Areas Protegidas de Uruguay'. The data provided correspond to 13 snapshots in the period of time from August 2008 to April 2011: Aug-2008, Sep-2008, Nov-2008, Jun-2009, Jul-2009, Sep-2009, Oct-2009, Dec-2009, Jan-2010, Feb-2010, May-2010, Jul-2010 and Apr-2011. Originally, 16 PAs (Protected Areas) had been defined and by April 2011 this number increased up to twenty. Those data have been experimenting changes not only in the values of the properties but also in the geometries. Implicitly, each consecutive pair of these snapshots defines a valid time interval. With all this information, we have processed these files and created a unique table, called 'Area'. A PA is characterized by several user-defined attributes (e.g. 'ID', 'nombre', 'límites', 'est_avance', 'decreto', etc.) plus the time interval ('from' and 'to' attributes) when this information was valid. In the table Area, each PA contains as many tuples as its different valid time intervals. The ID attribute of a PA is the only invariant one. If a PA, whose geometry is a polygon, it is divided into two, and a new tuple is generated with the same previous user-defined information but a multi-polygon shape instead.

id	name
1	Artigas
2	Canelones
3	Cerro Largo
4	Colonia
5	Durazno
6	Flores
7	Florida
8	Lavalleja
9	Maldonado
10	Montevideo
11	Paysandú
12	Río Negro
13	Rivera
14	Rocha
15	Salto
16	San José
17	Soriano
18	Tacuarembó
19	Treinta y Tres

Table 6.3: Deparment names of Uruguay

Figure 6.4 depicts a portion of how the evolution of PAs is stored in the database (table named Area). The special attributes 'from' and 'to' timestamp the valid interval of each PA. In the first stage, our Extract Transformation Load (ETL) process generated in the database a unique table of PAs with 231 tuples. If two time intervals exist for a single PA this means that at least one attribute differs in values, otherwise these tuples are coalesced. For example, the second, third and fourth tuples do not differ in anything except their time intervals which are consecutive. Thus, in the second state these tuples with intervals [2008-09-01, 2008-10-31], [2008-11-01, 2009-05-31] and [2009-06-01, 2009-06-30] have been coalesced into only one tuple with interval [2008-09-01, 2009-06-30]. More precisely, in the second stage, our ETL coalesced tuples and we obtained 68 tuples, as shown in Figure 6.5.

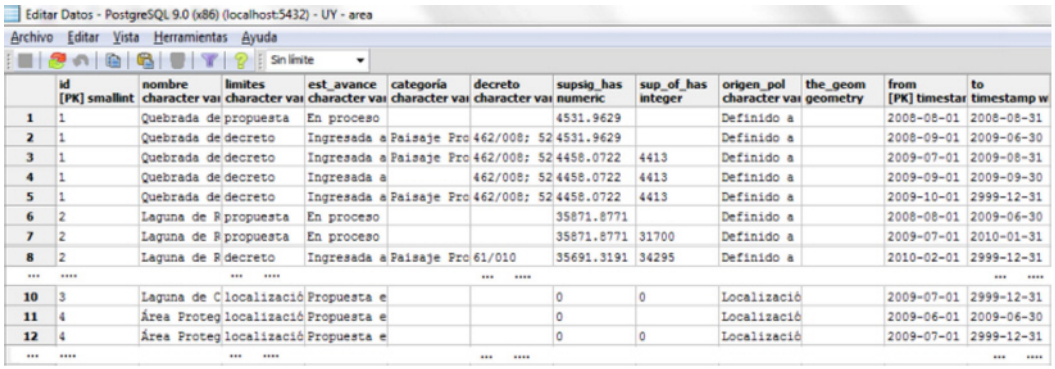

Editar Datos - PostgreSQL 9.0 (x86) (localhost:5432) - UY - area

Archivo Editar Vista Herramientas Ayuda

100 filas

	id [PK] sma	nombre character varying(80	limites character var	est_avance character varyir	categoria character varyir	decreto character var	supsig_has numeric	sup_of_has integer	origen_pol character var	the_geom geometry	from [PK] timestar	to timestamp w
1	1	Quebrada de los C	propuesta	En proceso de			4531.9629		Definido a		2008-08-01	2008-08-31
2	1	Quebrada de los C	decreto	Ingresada al :	Paisaje Prote	462/008; 52	4531.9629		Definido a		2008-09-01	2008-10-31
3	1	Quebrada de los C	decreto	Ingresada al :	Paisaje Prote	462/008; 52	4531.9629		Definido a		2008-11-01	2009-05-31
4	1	Quebrada de los C	decreto	Ingresada al :	Paisaje Prote	462/008; 52	4531.9629		Definido a		2009-06-01	2009-06-30
5	1	Quebrada de los C	decreto	Ingresada al :	Paisaje Prote	462/008; 52	4458.0722	4413	Definido a		2009-07-01	2009-08-31
6	1	Quebrada de los C	decreto	Ingresada al :		462/008; 52	4458.0722	4413	Definido a		2009-09-01	2009-09-30
7	1	Quebrada de los C	decreto	Ingresada al :	Paisaje Prote	462/008; 52	4458.0722	4413	Definido a		2009-10-01	2009-11-30
8	1	Quebrada de los C	decreto	Ingresada al :	Paisaje Prote	462/008; 52	4458.0722	4413	Definido a		2009-12-01	2009-12-31
9	1	Quebrada de los C	decreto	Ingresada al :	Paisaje Prote	462/008; 52	4458.0722	4413	Definido a		2010-01-01	2010-01-31
10	1	Quebrada de los C	decreto	Ingresada al :	Paisaje Prote	462/008; 52	4458.0722	4413	Definido a		2010-02-01	2010-04-30
11	1	Quebrada de los C	decreto	Ingresada al :	Paisaje Prote	462/008; 52	4458.0722	4413	Definido a		2010-05-01	2010-06-30
12	1	Quebrada de los C	decreto	Ingresada al :	Paisaje Prote	462/008; 52	4458.0722	4413	Definido a		2010-07-01	2011-03-31
13	1	Quebrada de los C	decreto	Ingresada al :	Paisaje Prote	462/008; 52	4458.0722	4413	Definido a		2011-04-01	2999-12-31
14	2	Laguna de Rocha	propuesta	En proceso de			35871.8771		Definido a		2008-08-01	2008-08-31
15	2	Laguna de Rocha	propuesta	En proceso de			35871.8771		Definido a		2008-09-01	2008-10-31
...	...	...	...	...	...	...	...				...	...
17	2	Laguna de Rocha	propuesta	En proceso de			35871.8771		Definido a		2009-06-01	2009-06-30
18	2	Laguna de Rocha	propuesta	En proceso de			35871.8771	31700	Definido a		2009-07-01	2009-08-31
19	2	Laguna de Rocha	propuesta	En proceso de			35871.8771	31700	Definido a		2009-09-01	2009-09-30
20	2	Laguna de Rocha	propuesta	En proceso de			35871.8771	31700	Definido a		2009-10-01	2009-11-30
21	2	Laguna de Rocha	propuesta	En proceso de			35871.8771	31700	Definido a		2009-12-01	2009-12-31
22	2	Laguna de Rocha	propuesta	En proceso de			35871.8771	31700	Definido a		2010-01-01	2010-01-31
23	2	Laguna de Rocha	decreto	Ingresada al :	Paisaje Prote	61/010	35691.3191	34295	Definido a		2010-02-01	2010-04-30
...	...	...	...	...	...	...	...				...	...

Figure 6.4: Table Area generated by ETL first stage


Editar Datos - PostgreSQL 9.0 (x86) (localhost:5432) - UY - area

Archivo Editar Vista Herramientas Ayuda

Sin límite

	id [PK] smallint	nombre character var	limites character var	est_avance character var	categoria character var	decreto character var	supsig_has numeric	sup_of_has integer	origen_pol character var	the_geom geometry	from [PK] timestar	to timestamp w
1	1	Quebrada de	propuesta	En proceso			4531.9629		Definido a		2008-08-01	2008-08-31
2	1	Quebrada de	decreto	Ingresada a	Paisaje Pro	462/008; 52	4531.9629		Definido a		2008-09-01	2009-06-30
3	1	Quebrada de	decreto	Ingresada a	Paisaje Pro	462/008; 52	4458.0722	4413	Definido a		2009-07-01	2009-08-31
4	1	Quebrada de	decreto	Ingresada a		462/008; 52	4458.0722	4413	Definido a		2009-09-01	2009-09-30
5	1	Quebrada de	decreto	Ingresada a	Paisaje Pro	462/008; 52	4458.0722	4413	Definido a		2009-10-01	2999-12-31
6	2	Laguna de R	propuesta	En proceso			35871.8771		Definido a		2008-08-01	2009-06-30
7	2	Laguna de R	propuesta	En proceso			35871.8771	31700	Definido a		2009-07-01	2010-01-31
8	2	Laguna de R	decreto	Ingresada a	Paisaje Pro	61/010	35691.3191	34295	Definido a		2010-02-01	2999-12-31
...	....		...	....		...	....					... .
10	3	Laguna de C	localizació	Propuesta e			0	0	Localizació		2009-07-01	2999-12-31
11	4	Área Proteg	localizació	Propuesta e			0		Localizació		2009-06-01	2009-06-30
12	4	Área Proteg	localizació	Propuesta e			0	0	Localizació		2009-07-01	2999-12-31
...	....		...	....		...	....					... .

Figure 6.5: Colaesced table Area generated by ETL second stage

6.2.3 SOLAP Cube

We built a Weather SOLAP data cube using real information downloaded from Instituto Nacional de Estadistica in Uruguay[5].

Climatic information was provided by the 'Dirección Nacional de Meteorología'. The Weather cube contains two measures: median temperature (in Celsius degrees) and precipitation (in l/m^2), summarized by weather stations in different cities and months, corresponding to the period [2009, 2010]. Thus, the Weather cube was modeled with the following two dimensions:

[5]http://www.ine.gub.uy/biblioteca/anuario2011/Anuario 2011.pdf

- *Time Dimension.* It contains the following level hierarchy: month → year → All
- *Spatial Dimension.* It has non-geometric objects, i.e. the spatial levels contain text (name of the cities and states). It contains the level hierarchy: city → department → All.

6.3 Summary

We have presented a case study were we have integrated real data of Static GIS (Functional Ecosystem Types, Ornito-geo zones and Deparment names of Uruguay), Temporal GIS (PAs: 13 snapshots in the period of time from August 2008 to April 2011) and SOLAP data cube (Median temperature and Precipitation in [2009, 2010]). In **Chapter 7**, we will show how our proposal can be applied, by developing different TPiet-QL queries that mix all kind of mentioned objects.

Chapter 7

TPiet-QL By Example

In this chapter we show different kinds of queries in order to have an overview of TPiet-QL. These queries are also available through the website[1].

7.1 Querying Spatio-Temporal Objects

TPiet-QL is specifically designed to use predicates over spatio-temporal objects, intervals, and instants. In order to get some insight of the language and user interface, we propose some examples. We will use the objects in table Area, i.e. PAs.

Query 1: PAs at a specific date - *Show PAs that existed on July 5, 2009 with the following information at the time: ID, the geom, est avance and time interval.*

```
SELECT GIS SNAPSHOT id, est_avance, the_geom,
           commoninterval() AS interval
FROM area
WHERE at(area, '2009-07-05')
```

Note that although we explicitly recover the commoninterval() for the objects that satisfy the condition, we use the snapshot clause for assuring that the information is being displayed in only one map. The result set contains only 18 of the 20 PAs existent at that moment, since the other two were created later as we can see in Figure 7.1.

We can also query without `SNAPSHOT` to obtain this information:

```
SELECT GIS id, est_avance, the_geom,
        commoninterval() AS interval
FROM area
WHERE at(area, '2009-07-05')
```

[1] http://www.fing.edu.uy/inco/proyectos/laccirPiet/stpietqlweb/

Figure 7.1: Query 1 - PAs at a specific date

Nevertheless, the information is always visualized in a temporal partition. For example, the PAs with ID 1 and 3 existed at that moment, their valid intervals were *[2009-07-01, 2009-08-31]* and *[2009-07-01, 2999-12-31]*, respectively. That is, to assure that we see all the information over only one map, we have to configure the user interface with as many Months per Map as months passed since the smallest 'from' of these intervals. In order to avoid this, it is more practical to use the `SNAPSHOT` clause.

Query 2: Approximating PAs by MBRs - *For each current PA, show its minimum bounded box, its shape and its progress status.*

```
SELECT GIS SNAPSHOT the_geom, envelope(the_geom) AS MBR,
        the_geom, est_avance, nombre
FROM area
WHERE to = Now()
```

Note that if it is not necessary to visualize the exact geometry of a PA it is better to simplify its shape, for example replacing its geometry by a rectangle. In several situations, browsers are not capable of displaying complex geometries. In our dataset the number of points of the polygons of PAs range from 35 to 2890. Problems can appear when the browsers need to display many of these complex polygons altogether (note that for visualizing all the PAs that evolved throughout time the browser might manage 39892 points). Figure 7.2 shows Query 2 resultset.

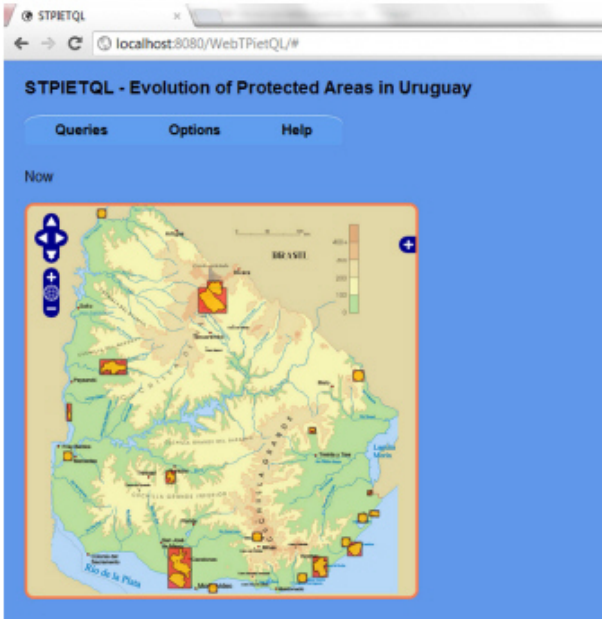


Figure 7.2: Query 2 - Approximating PAs by MBRs

Query 3: Proximity - *Show PAs that coexisted (overlapped in time) and were very close at each other, suggesting that they can be eventually extended and merged into a single one, or grouped together for administrative purposes.*

```
SELECT GIS a1.ID, a2.ID, a1.nombre, a2.nombre,
        commoninterval() as interval, a1.the_geom, a2.the_geom,
        distance(a1.the_geom, a2.the_geom) as dist
FROM OVERLAP area AS a1, area AS a2
WHERE a1.id < a2.id
    AND distance(a1.the_geom, a2.the_geom) < 500
```

In this query, we need to use the *Overlap Join*, since we are interested in temporal coexistence. Moreover, we want to see both close geometries (distance less than 500 m) in the same map for distance comparison. Each time we use the *geometry* data type in a `SELECT` clause, a new layer is created in the map. Here we have two geometries: *a1.the geom* and *a2.the geom*, so we have two layers for visualizing. We can choose different colors for each of them as we can see in Figure 7.3. Finally, during the first period (left map) only two pairs of PAs satisfied the queries. From the second period, three red/green 26 pairs of PAs are displayed, i.e. PAs (6, 22), (20, 37) and (3, 12). The first two are adjacent (distance 0), the latter are very close (distance 421 m).

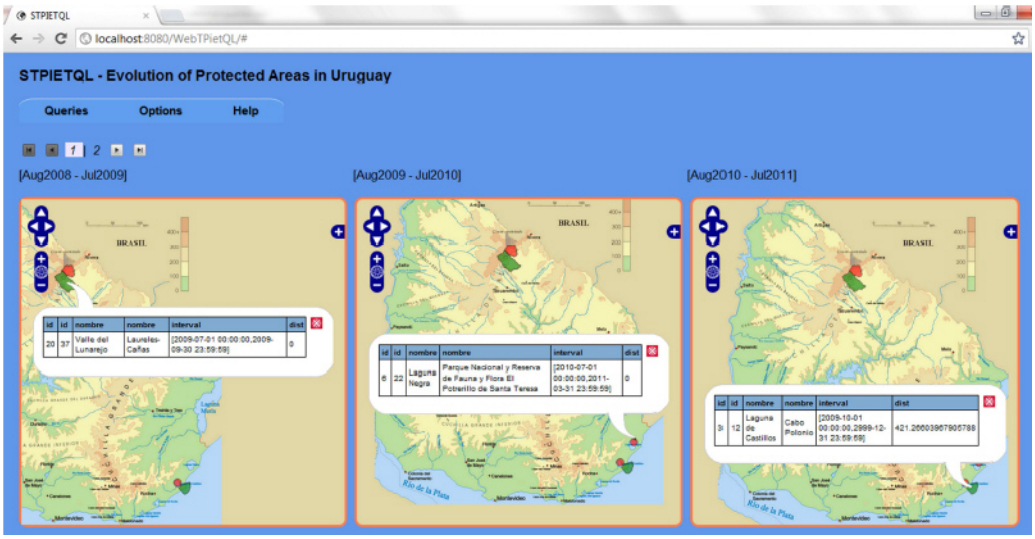


Figure 7.3: Query 3 - Proximity

7.2 Querying Spatio-Temporal Objects and Static Objects

We show now how to mix spatio-temporal objects with static objects in our language. TPiet-QL engine detects that a layer is static if its underlying table does not contain the special 'to' and 'from' attributes. Moreover, these attributes cannot be explicitly used in the filter (`WHERE`) clause. Nevertheless, when using a static GIS layer l in TPiet-QL, its lifespan matches with an open time interval, i.e. $\mathcal{L}(l)$=(-∞,∞), which means that is always valid. Thus, the semantics is straightforward.

If we mix temporal and static objects only temporal ones are consider for calculating their temporal overlapping, since (-∞,∞) is the identity interval for the intersection operator. If an expression only contains static objects, 'to' and 'from' attributes cannot be used, but instead the *CommonInterval()*, *CMStart()* and *CMEnd()* functions can be selected. TPiet-QL engine translates -∞ and ∞ values to very small and big numbers, respectively. By default, those values are 1000 and 2999, but can be changed in the configuration file. Now, we will list some queries that use not only Area, but static objects in Department, TFE and ornitogeo tables.

Query 4: Departments and their PAs - *For each department (name and geometry) show the information of its current PAs, expressed in the form (ID, name, geometry) only in one map.*

```
SELECT GIS SNAPSHOT name AS deptname, geom AS deptgeom, ID, the_geom,
        area.nombre, commoninterval() as interval
FROM OVERLAP area, department
WHERE to = Now() AND intersects(geom, the_geom)
```

Only 15 of the 21 departments contain PAs as shown in Figure 7.4

Figure 7.4: Query 4 - Departments and their PAs

Query 5: PAs fully contained in one TFE - *For each PA that is totally contained in one of the size TFEs, show its exact geometry, ID and name joint with the type and geometry of the corresponding TFE.*

```
SELECT GIS the_geom, ID, nombre, tipo_func, TFE.geom
FROM area, TFE
WHERE contains(geom, the_geom)
```

The result set can be visualized in Figure 7.5. We have hidden the Uruguay Map for better visualizing. We zoomed first and second maps to show that in the latter a new PA appeared (ID= 22) and then its corresponding TFE is also displayed with it.

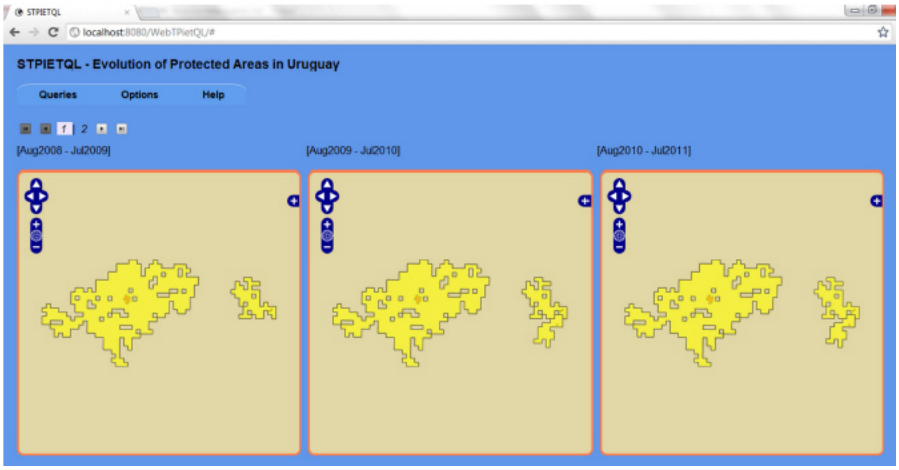


Figure 7.5: Query 5 - PAs fully contained in one TFE

Query 6: Huge departments with at least one specific ornitogeo zones - *Show huge departments, with surface at least 640 Km^2, such that intersects at least one of these two ornitogeo zones: Sabana esteparia serrana o Pradera*

```
SELECT GIS d.geom, d.name, o.geom, o.unidad,
    commoninterval() as interval
FROM OVERLAP department AS d , ornitogeo AS o
WHERE Area(o.geom) > 640000000 AND
    intersects(o.geom, d.geom) AND
    (o.unidad = 'Sabana esteparia serrana' OR o.unidad = 'Pradera')
```

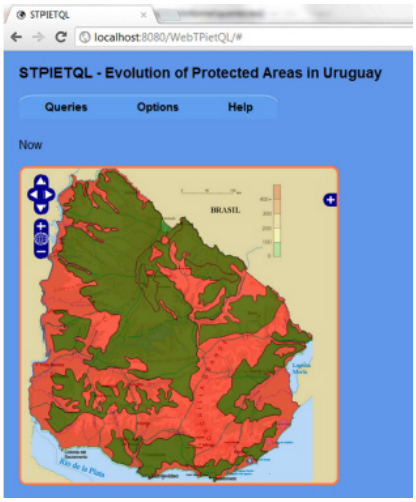


Figure 7.6: Query 6 - Huge departments with at least one specific ornitogeo zones

The result set can be visualized in Figure 7.6. Only two PAs satisfied this condition.

7.3 Querying Static Objects, Spatio-Temporal Objects and SOLAP cubes

Now we will get more involved and show TPiet-QL that mix all kind of objects, i.e. static objects, spatio-temporal ones and spatial entities that belong to SOLAP cubes.

Query 7: PAs, departments and precipitation - *Show PAs and the name of the department that contains them, only for departments which had a accumulated precipitation not exceeding the limit of 2800 l/m^2.*

```
SELECT GIS the_geom, ID, nombre, department.name
FROM area, department
WHERE contains(department.geom, the_geom)
AND department.name IN
        (SELECT CUBE filter([Geography].[Department].Members,
            [Measures].[Precipitation] < 2800)
            FROM [Weather] )
```

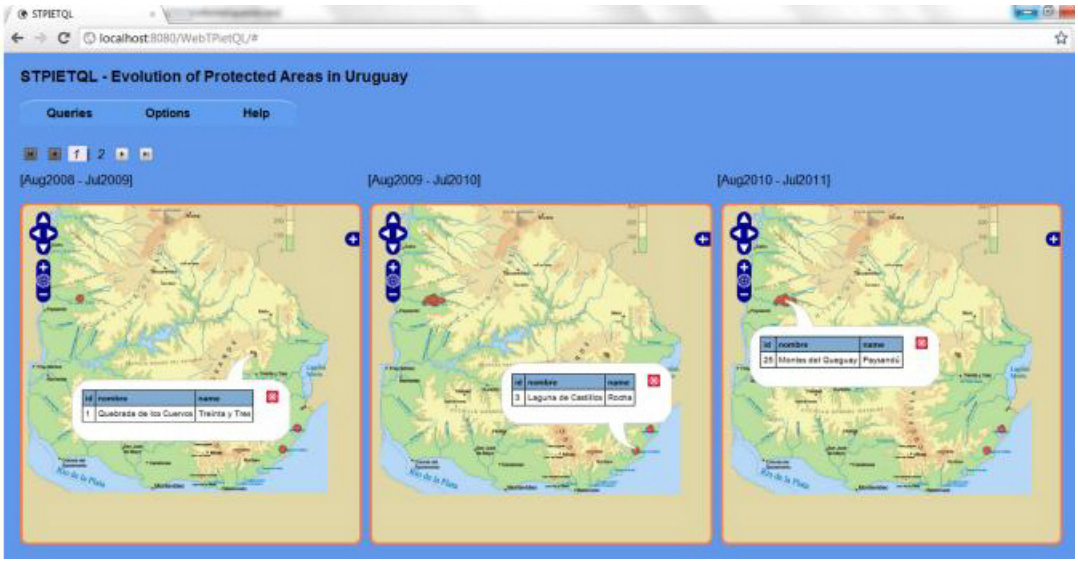


Figure 7.7: Query 7 - PAs, departments and precipitation

If we would executed only the first part of the query, i.e. without restricting the GIS results by a SOLAP cube, 6 departments would be returned: *Cerro Largo, Lavalleja, Paysandú, Río Negro, Rocha and Treinta y Tres.* The nested SOLAP query, which calculates departments summarizing precipitation with values less than 2800 l/m^2, showed the following results: *Colonia (2,286.30 ml), Maldonado (2,283.60 ml), Paysandu (2,796.10 ml), Rocha (2,626.40 ml), Salto (2,781.60 ml) and Treinta y Tres (2,582.50 ml).*

Finally, PAs contained by departments that satisfy the SOLAP condition are those which are contained in *Paysandú, Rocha* and *Treinta y Tres*. Notice that *Río Negro, Lavalleja* and *Cerro Largo* were not recovered since they only satisfied the GIS condition. Same criteria is applied with *Maldonado, Colonia* and *Salto*, since they were not recovered because they only satisfied the SOLAP condition. Figure 7.7 depicts the result set.

Query 8: PAa changed their shape with a buffer around within department with specific precipitation - *Show PAs, throughout time, whose new geometry contains the previous one or viceversa. Exclude such geometries that are the same (strict case of containment). Restrict to those PAs that intersects some department with accumulative precipitation was greater than 550 ml during period from second semester of year 2009 and first semester of year 2010.*

The query can be expressed in either of these two syntax (ST_Contains is the inverse of ST_Within):

Syntax 1:

```
SELECT GIS a1.id, a1.nombre, area(a1.the_geom) as area1,
    area(a2.the_geom) as area2,
    a1.the_geom, a2.the_geom
FROM area AS a1, area AS a2, department
WHERE a1.id = a2.id
AND NOT ( StartsBefore(a2, a1.to) )
AND st_within(a2.the_geom, a1.the_geom)
AND NOT ( st_equals(a1.the_geom, a2.the_geom ) )
AND st_intersects(geom, a2.the_geom)
AND department.name IN
            (SELECT CUBE Filter([Geography].[Department].Members,
                Precipitation < 800)
            FROM [Weather]
            SLICE {[TimeDimension].[Semester].[S2-2009],
                [TimeDimension].[Semester].[S1-2010]})
```

Syntax 2:

```
SELECT GIS a1.id, a1.nombre, area(a1.the_geom) as area1,
    area(a2.the_geom) as area2,
    a1.the_geom, a2.the_geom
FROM area AS a1, area AS a2, department
WHERE a1.id = a2.id
AND NOT ( StartsBefore(a2, a1.to) )
AND st_contains(a1.the_geom, a2.the_geom)
AND NOT ( st_equals(a1.the_geom, a2.the_geom ) )
AND st_intersects(geom, a2.the_geom)
AND department.name IN
        (SELECT CUBE Filter([Geography].[Department].Members,
            Precipitation < 800)
        FROM [Weather]
        SLICE {[TimeDimension].[Semester].[S2-2009],
            [TimeDimension].[Semester].[S1-2010]})
```

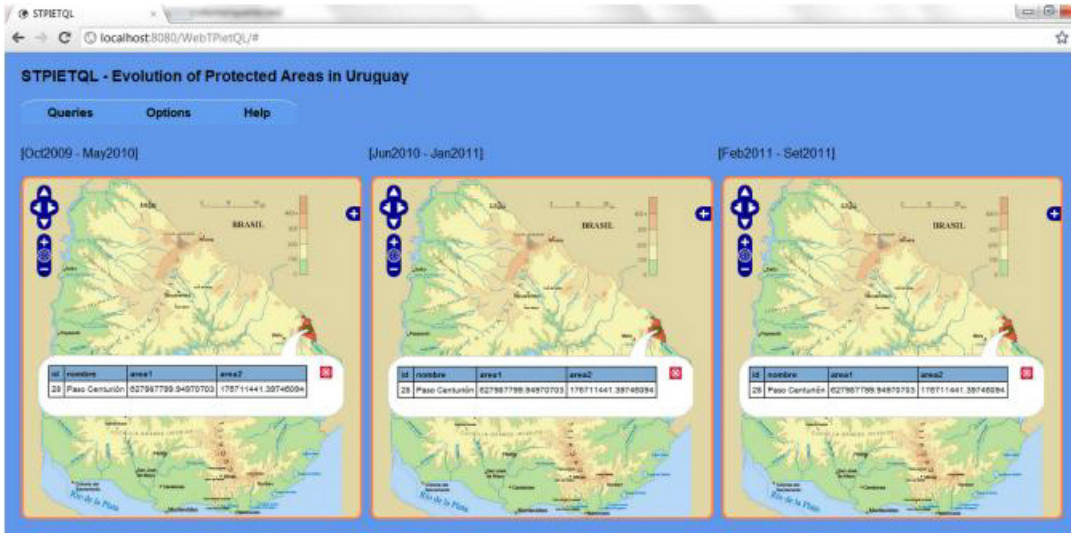


Figure 7.8: Query 8: PAa changed their shape with a buffer around within department with specific precipitation

The SOLAP cube returns the following departments: *Cerro Largo, Colonia, Flores, Maldonado, Rocha* and *Tacuarembó*. The only PA that satisfies the GIS condition is *Paso Centurión* which intersects the *Cerro Largo* department. Figure 7.8 shows the result set.

Query 9: Recently created PAs within specific climatic changes - *Recently created PAs, i.e. created since 2010, that interact with departments whose average of temperature considering firsts semesters and seconds semesters show a decrease that range from (18.7, ∞) to (-∞, 15) Celsius degree. Show them with a buffer around of 5 Km (show also the exact geometry) grouped by semesters.*

```
SELECT GIS a1.id, a1.nombre,
    buffer(a1.the_geom, 5000) as buffer, a1.the_geom
FROM area as a1, department
WHERE NOT (a1.id IN
        (SELECT GIS a2.id
            FROM area AS a2
            WHERE startsbefore(a2, '2010-01-01 00:00:00'))
        )
AND st_intersects(geom, the_geom)
AND department.name IN
    (SELECT CUBE Filter([Geography].[Department].Members,
        Temperature < 15)
    FROM [Weather]
    SLICE {[TimeDimension].[Semester].[S2-2009],
        [TimeDimension].[Semester].[S2-2010]}
    )
AND department.name IN
    (SELECT CUBE Filter([Geography].[Department].Members,
        Temperature > 18.7)
    FROM [Weather]
    SLICE {[TimeDimension].[Semester].[S1-2009],
        [TimeDimension].[Semester].[S1-2010]})
```

First SOLAP query returns only *Rocha* department. The second SOLAP query returns all departments except *Canelones, Flores, Florida, Lavalleja* and *Soriano*. Thus, only *Rocha* experimented this climatic change. PAs whose IDs are 22 and 67 are the only ones that appeared since 2010. Moreover the former intersects *Rocha* department and the latter *Artigas*. We obtain only the PA with ID 22 in the result set. We can see that its name changed thoughout this year. More precisely, during the first semester was named *'Parque Nacional y Reserva de Fauna y Flora El Potrerillo de Santa Teresa'* and during the second Semester *'Potrerillo de Santa Teresa'*.

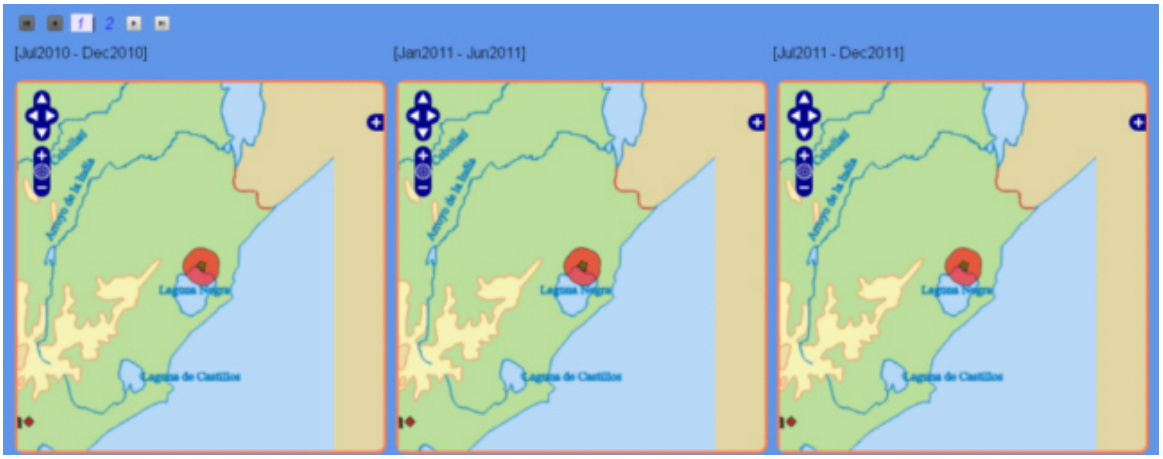


Figure 7.9: Query 9: Recently created PAs within specific climatic changes

7.4 Summary

We have developed different TPiet-QL queries that were designed to use predicates over spatio-temporal objects, intervals, and instants. Then, we showed how to integrate those spatio-temporal objects with static objects in our language, finishing with some examples that model how TPiet-QL interacts with a mix of the three kind of objects mentioned before: static objects, spatio-temporal ones and spatial entities that belong to SOLAP cubes.

Chapter 8

Conclusion and Open Research Directions

TPiet, a spatio-temporal OLAP was proposed based on Piet data model. The extension of its architecture required several technological enhancements to support discrete changes. Protected areas were taken as the case study and a world-real dataset was used; it was obtained from the collaborative work with National Protected Area System of Uruguay. Moreover, because of our data model flexibility, is able of mixing temporal, static information and OLAP data cubes in the same framework, we have enriched the case study with ecosystem data, ornito-ecological zones, administrative division of the country and climatic environment information, all of them obtained from real public data. Besides the data model, a powerful query language which manages spatial and temporal capabilities, offering the possibility of restricting temporal GIS queries with SOLAP conditions, was proposed and developed.

The TPiet-QL programmatic API was implemented and was included in the LACCIR[1] project R1210LAC004 - *Monitoring Protected Areas using an OLAP-enabled Spatio-Temporal Geographic Information System*[2], as part of the Web Graphic Interface[3] (A Graphic Interface designed with the aim of helping analysts to compare temporal data evolution). Many technical problems were faced at project start-up, one of the biggest being the dynamic visualization of geographic information, in the form of a SQL query result, run from a web-browser. Our first approach pointed to MapServer[4], a popular platform for building web mapping applications. MapServer is mainly based on a configuration file called *mapfile*, that contains the data source, map styling and server directives. In order to integrate it with PostGIS, the accessed table, the column with spatial data and the selection conditions must be specified separately in the *mapfile*. The results are automatically displayed by MapServer, with no possibility of formatting nor editing them beyond the available features through the *mapfile*. We built a prototype that accomplished

[1] http://www.laccir.org
[2] http://www.fing.edu.uy/inco/proyectos/laccirPiet/
[3] http://www.fing.edu.uy/inco/proyectos/laccirPiet/stpietqlweb/
[4] http://mapserver.org

the objective of showing dynamic geometric query results, by building an ad-hoc *mapfile* on the fly for each executed query. This required a query string manipulation to fit the *mapfile* format. We also analyzed another GIS system for web applications called OpenLayers[5], a javascript library with no server-side dependencies, which met our needs in a simpler way. OpenLayers allowed the segregation of query execution and the result displaying, fitting better with our requirements. By simply converting the SQL query result into JSON format, we could send it to OpenLayers and display all the information in HTML elements, gaining more control on the processing steps, along with an easier way to display the related information. So we put our efforts in developing a pilot with this technology. Once finished, we were able to select OpenLayers to build the final solution for the project. A direction for future work is related to the visualization of features that are totally overlapped in the same layer and remain inaccessible. A possible solution is to add capabilities to move the overlapped layers under user requirements forwards and backwards, allowing the layer with the desired information move up to front. Another solution consists of displaying inside the same popup window not only the information of the top layer, but also the information of all overlapped layers, i.e. a row for each layer clicked directly or indirectly.

The support of raster data in the language, and a study of how TPiet-QL can be used to allow the definition of constraints, along the lines of [21], [6] (the STRM model commented in **Chapter 1** includes a language to define spatio-temporal integrity constraints), is an open problem that still needs to be investigated.

Given the relatively small number of protected areas at this time, the prototype is able to run in reasonable execution times, showing real data. However, larger amounts of data will require more sophisticated query processing and optimization. This constitutes the next steps of our future work.

[5]http://openlayers.org

Bibliography

[1] James F. Allen. Maintaining Knowledge about Temporal Intervals. *Commun. ACM*, 26(11):832–843, 1983.

[2] Eduardo Arballo and Jorge L. Cravino. *Aves del Uruguay. Manual Ornitologico. Struthioniformes a Gruiformes*. Hemisferio Sur, 1999.

[3] Santiago Baeza, José M Paruelo, and Alice Altesor. Caracterización funcional de la vegetacion del Uruguay mediante el uso de sensores remotos. *Interciencia*, 31(5):7, May 2006.

[4] Yvan Bédard, Sonia Rivest, and Marie-Josée Proulx. Spatial Online Analytical Processing (SOLAP): Concepts, Architectures, and Solutions from a Geomatics Engineering Perspective. *Data Warehouses and OLAP: Concepts, Architectures, and Solutions*, pages 298–319, 2007.

[5] Pablo Bisceglia, Leticia I. Gómez, and Alejandro A. Vaisman. Temporal SOLAP: Query Language, Implementation, and a Use Case. In *AMW*, pages 102–113, 2012.

[6] Sophie Cockcroft. A taxonomy of Spatial Data Integrity Constraints. *Geoinformatica*, 1(4):327–343, December 1997.

[7] Johann Eder, Christian Koncilia, and Tadeusz Morzy. The COMET Metamodel for Temporal Data Warehouses. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, CAiSE '02, pages 83–99, London, UK, UK, 2002. Springer-Verlag.

[8] M. J. Egenhofer. Spatial SQL: A Query and Presentation Language. *IEEE Trans. on Knowl. and Data Eng.*, 6(1):86–95, February 1994.

[9] Leticia I. Gómez, Sophie Haesevoets, Bart Kuijpers, and Alejandro A. Vaisman. Spatial aggregation: Data model and implementation. *Inf. Syst.*, 34(6):551–576, 2009.

[10] Leticia I. Gómez, Alejandro A. Vaisman, and Sebastián Zich. Piet-QL: a query language for GIS-OLAP integration. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, GIS '08, pages 27:1–27:10, New York, NY, USA, 2008. ACM.

[11] Ralf Hartmut Guting, Victor Almeida, Dirk Ansorge, Thomas Behr, Zhiming Ding, Thomas Hose, Frank Hoffmann, Markus Spiekermann, and Ulrich Telle. SECONDO: An Extensible DBMS Platform for Research Prototyping and Teaching. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 1115–1116, Washington, DC, USA, 2005. IEEE Computer Society.

[12] Carlos A. Hurtado and Alberto O. Mendelzon. OLAP dimension constraints. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 169–179, New York, NY, USA, 2002. ACM.

[13] Ralph Kimball. *The data warehouse toolkit: practical techniques for building dimensional data warehouses*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[14] Alberto O. Mendelzon and Alejandro A. Vaisman. Temporal Queries in OLAP. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 242–253, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[15] Nikos Pelekis, Yannis Theodoridis, Spyros Vosinakis, and Themis Panayiotopoulos. Hermes - a framework for location-based data management. In *Proceedings of the 10th international conference on Advances in Database Technology*, EDBT'06, pages 1130–1134, Berlin, Heidelberg, 2006. Springer-Verlag.

[16] Nikos Pelekis, Babis Theodoulidis, Ioannis Kopanakis, and Yannis Theodoridis. Literature review of spatio-temporal database models. *Knowl. Eng. Rev.*, 19(3):235–274, September 2004.

[17] Sonia Rivest, Yvan Bédard, and Pierre Marchand. Towards better support for spatial decision-making: Defining the characteristics of spatial on-line analytical processing (SOLAP). In *Geomatica*, pages 539–555, 2001.

[18] Abdullah Uz Tansel, James Clifford, Shashi Gadia, Sushil Jajodia, Arie Segev, and Richard Snodgrass, editors. *Temporal databases: Theory, Design, and Implementation*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1993.

[19] David Toman. Point vs. interval-based query languages for temporal databases (extended abstract). In *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '96, pages 58–67, New York, NY, USA, 1996. ACM.

[20] N. Tryfona and T. Hadzilacos. Logical Data Modeling of SpatioTemporal Applications: Definitions and a Model. In *Proceedings of the 1998 International Symposium on Database Engineering & Applications*, IDEAS '98, pages 14–23, Washington, DC, USA, 1998. IEEE Computer Society.

[21] Nectaria Tryfona and Christian S. Jensen. Conceptual Data Modeling for Spatiotemporal Applications. *Geoinformatica*, 3(3):245–268, 1999.

[22] Monica Wachowicz and Richard Healey. Towards temporality in GIS. *Innovations in GIS. Ed. Michael F. Worboys*, 1:105–115, 1994.

[23] Michael F. Worboys. A model for spatio-temporal information. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 602–611, Charleston, South Carolina, 1992.