# Temporal SOLAP: Query Language, Implementation, and a Use Case

Pablo Bisceglia[1] and Leticia Gómez[2] and Alejandro Vaisman[3]

[1] Universidad de Buenos Aires, Argentina pbisceglia@gmail.com
[2] Instituto Tecnológico de Buenos Aires, Argentina lgomez@itba.edu.ar
[3] Université Libre de Bruxelles, Belgium avaisman@ulb.ac.be

**Abstract.** The integration of Geographic Information Systems (GIS) and On-Line Analytical Processing (OLAP), denoted SOLAP, is aimed at exploring and analyzing spatial data. In real-world SOLAP applications, spatial and non-spatial data are subject to changes. In this paper we present a temporal query language for SOLAP, called TPiet-QL, supporting so-called discrete changes (for example, in land use or cadastral applications there are situations where parcels are merged or split). TPiet-QL allows expressing integrated GIS-OLAP queries in an scenario where spatial objects change across time. We also present a prototype implementation, and show how this application is used in a real-world scenario: the analysis of protected areas in Uruguay.

## 1 Introduction and Problem Statement

In Geographic Information Systems (GIS), spatial data are organized in thematic layers, stored in suitable data structures, while associated attributes are usually stored in conventional relational databases. On the other hand, OLAP (On-Line Analytical Processing) [2] provides a set of tools and algorithms that allow efficiently querying multidimensional repositories called Data Warehouses (DW). OLAP data are organized as a set of *dimension hierarchies* and *fact tables*, and can be perceived as a *data cube*, where each cell contains a measure or set of measures of interest. The problem of integrating OLAP and GIS systems for decision-making analysis has been called SOLAP [4].

In real-world applications, spatial objects in a layer can be added, removed, split, merged, or their shape may change. Tryfona and Jensen [1], classify spatio-temporal applications according with the kind of support of the changes occurring in spatial objects. They distinguish 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* (e.g., a stain in a river).

Piet [3], is a data model proposed for SOLAP. In this paper we present an extension to Piet-QL (a query language associated with Piet) that supports *discrete changes,* and show a prototype implementation applied to a real-world

scenario: the analysis of protected areas in Uruguay[4]. The International Union for Conservation of Nature (IUCN) defines a protected area as "..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". The World Database on Protected Areas (WDPA), a joint venture between the United Nations Environment Programme and the IUCN, contained 147,897 protected areas in July 2010. Protected areas evolve across time: new areas are added to the system, management plans are implemented, and the administrative status of each area changes. This behavior is sketched in Figure 1. Here, a protected area denoted PA 1 has been created at time $t = 1$ and is in the 'In-process' status. There is also another protected area denoted PA 2, also with status 'In-process'. At $t = 2$, the shape of PA 1 has changed, and its area increased. Finally, at $t = 4$, PA 1 is even larger, and now the status has changed to 'Confirmed'. At $t = 3$, another area, PA 3 has been created. It changed its shape at $t = 4$. Information about protected areas can be stored in a conventional DW, where each spatial object representing an area has a corresponding element in a dimension Geography, e.g., with a hierarchy where areald aggregates over department, which in turn aggregates over region. Other dimensions for analysis can exist, along with associated fact tables. In a *discrete changes* scenario like this, we could display the history of PA 1, or pose queries like "Protected areas where the average precipitation increased since 2008, for areas located at less than 100km from Montevideo". This requires extending non-temporal SOLAP data models and query languages (like Piet-QL) with temporal capabilities.

The paper is organized as follows. After an overview of related work (Section 2), we define the temporal data model (Section 3), and present the syntax and semantics of TPiet-QL in Section 4. We show a prototype implementation and the protected areas case study in Section 5, concluding in Section 6.

## 2 Related Work

Rivest *et al.* [4] 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. Piet [3] is a formal model for SOLAP, where the integration between GIS and OLAP is materialized through a function that maps elements in the DW to elements in the GIS layers. Piet comes equipped with a query language, Piet-QL, that supports the operators proposed by the Open Geospatial Consortium[5] for SQL, adding the necessary syntax to integrate OLAP operations through a language based on MDX[6]. Piet-QL supports (besides standard spatial and OLAP queries), spatial

[4] This use case is part of the project "Monitoring Protected Areas using an OLAP-enabled Spatio-temporal GIS" (http://piet.exp.dc.uba.ar/laccir), funded by LAC-CIR(http://www.laccir.org.)

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

[6] http://msdn2.microsoft.com/en-us/library/ms145506.aspx.

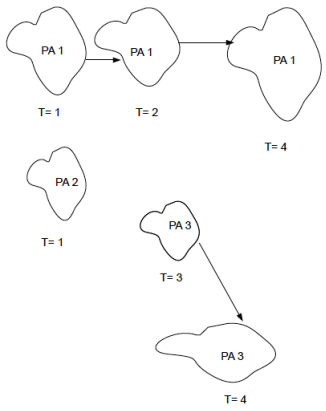


**Fig. 1.** Evolution of protected areas across time.

queries filtered using OLAP conditions, and OLAP queries filtered by spatial conditions. Filtering is implemented through a predicate denoted `IN`. For example, the query "Areas crossed by the 'Uruguay' river, with precipitation greater than 100 mm per year" reads in Piet-QL.

```
SELECT GIS ar.id
FROM pArea ar, rivers ri
WHERE intersects(ar,ri) AND ri.name = "Uruguay" AND ar IN(
   SELECT CUBE filter([Geography].[Geography areaId].Members,
   [Measures].[precipitation] > 100)
   FROM [Weather]);
```

Here, 'pArea' and 'rivers' represent two layers containing spatial objects (the protected areas and the rivers, respectively). There is also a data cube denoted Weather, such that its Geography dimension is linked to the 'pArea' layer in a way such that there is a function mapping spatial objects in the latter to members in the areaId dimension level. The OLAP subquery (identified with the keyword `CUBE`) is linked to the outer query (a spatial query identified with the keyword `GIS`) by the predicate `IN`. This OLAP subquery returns a collection of identifiers of spatial objects that satisfy it. The data model and the formal query language underlying TPiet-QL are discussed in [12].

The Spatio-Temporal Relational data Model (STRM), introduced by Tryfona and Hadzilacos [5], provides a set of constructs consisting in relations, layers, virtual layers, object classes, and constraints, all with spatial and temporal extent, on top of well-known models. In this model, a *layer* is a set of geometric figures like points, lines, regions or combinations of them, with associated values. A layer algebra, based on four operations over layers, provides a semantics to SOLAP.

## 3 Spatio-Temporal Piet

In the temporal extension to Piet, each tuple in a spatial relation is timestamped with its validity interval. Time is introduced as a new sort or domain (we work with interval-based domains). In temporal databases, the notions of *valid* and (*transaction*) time refer to the instants when data are valid in the real world, and recorded in the database, respectively [6]. We assume *valid* time support in this work. A distinguished variable *Now* represents the (moving) current time instant. The *lifespan* of a GIS layer $L$, *lifespan*$(L)$, is the collection of all the time instants where the layer is valid. The *lifespan* of a set of layers $\mathcal{L}$, $lifespan(\mathcal{L})$, is the union of the lifespans of all the layers in $\mathcal{L}$.

Given the above, a *Temporal GIS-OLAP Dimension Schema* $\text{TG}_{sch}$ is a tuple $\langle H, \mathcal{A}, \mathcal{D}, \mu \rangle$, where $H$ is a mapping from layers to geometries, $\mathcal{A}$ is a set of *partial* functions *Att* that map attributes in OLAP dimensions to GIS layers, $\mathcal{D}$ is a set of dimension hierarchies [7], and $\mu$ a dimension level in an OLAP Time dimension. Elements in $\mu$ belong to the temporal domain. Further, $H$, $\mathcal{A}$ 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 $L$, 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. Associated with a dimension schema, we have a dimension instance, which consists in: 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 $t$; A collection of functions $\alpha$ that map elements in OLAP dimension levels to geometric elements in a GIS layer, at a given time; A collection of dimension instances, one for each dimension schema $D \in \mathcal{D}$ in $\text{TG}_{sch}$. We assume that spatial objects have the same attributes throughout their lifespan.

The *data structure* of TPiet-QL consists of: (a) The DW structure. Contains dimension and fact tables. (b) The data structure for the map layers (one table per layer). Temporal attributes FROM and TO indicate the interval of validity of each object in a layer. (c) GIS-OLAP mapping information. Stores the relationship between geometric and application information (i.e., the $\alpha$ functions), including the interval of validity of each mapping.

When new spatial objects are created on the GIS side, the corresponding objects must be inserted in the DW 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. If an insertion on the GIS occurs without mapping the new object to a DW object, incomplete answers may be obtained, due to such incomplete mapping. In this paper we do not support temporal DWs (e.g., [8, 9]), i.e., dimensions are *static*, and only the current state of dimension data is available.

## 4 TPiet-QL: a Temporal SOLAP Query language

The main element in TPiet-QL is the *spatio-temporal object* defined next.

| | |
|---|---|
| `StartsBefore(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`. |
| `BeginsAfter (g,t)`: $\mathcal{G} \times T \rightarrow$ *boolean*; Given a spatio-temporal object and an instant, returns *True* if $t <$ `g.FROM`. | `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` $t >=$ `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`. |

**Fig. 2.** Predicates over spatio-temporal objects, intervals, and instants.

**Definition 1 (Spatio-temporal object).** *We denote by spatio-temporal object a tuple of the form* $\langle$*objectId*, *geometry*, *attribute*$_1$, ..., *attribute*$_n$, *interval*$\rangle$, *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* $[FROM, TO]$. □

In Definition 1, *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 paper we work with single intervals instead of temporal elements. This makes the paper easier to read, without reducing its substance. In what follows we refer to spatio-temporal objects as 'objects', and denote $\mathcal{G}$ a collection of spatio-temporal objects. Based on Allen's interval set of predicates [10], in Figure 2 we specify the syntax and semantics of a collection of predicates over spatio-temporal objects, intervals, and time instants.

*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 [6], 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. 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). An example (following Allen) is `before-join(X,Y)`, with condition $X.TO \leq Y.FROM$. Let us consider now a predicate $P_a$, specifying the equality of a collection of non-temporal attributes. A GT-join (standing for generic temporal join) corresponds to the expression $\sigma_{P_a \wedge overlap-join(X,Y)}(X, Y)$. Thus, given two

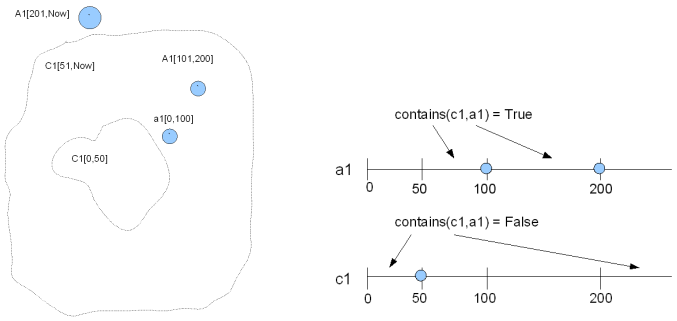


**Fig. 3.** A city and its airport (left); Interaction of $a_1$ and $c_1$ along their timelines (right)

tuples, the result of a GT-join are the tuples that have overlapping time intervals and satisfy the non-temporal predicate $P_a$. In the presence of spatio-temporal objects, the GT-join can be defined using the standard topological relationships [11], like `Touches`$(g_1, g_2)$, or `Contains`$(g_1, g_2)$. Consider two layers storing the histories of airports and cities. Figure 3 (left) shows two states of city $c_1$, in the intervals [0,50], and [51,Now]. Airport $a_1$ was first relocated at instant 100, and then, due to the city expansion, it was located outside the new city limits. Figure 3 (right) shows how the two objects $a_1$ and $c_1$ interact along their timelines: the airport is within the city limits in the intervals [51,100] and [101,200]. The relational representations are:

| 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 | 100 |
| a1 | g2 | ... | 101 | 200 |
| ... | ... | ... | ... | ... |

The query "pairs city-airport such that an airport was within the city limits" is the GT-join shown below. The result contains the tuples $\langle a1, c1, 51, 100\rangle$ and $\langle a1, c1, 101, 200\rangle$, representing (see Figure 3) that between instants 51 and 200, a1 remained within the city limits of c1.

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

*The TPiet-QL Query Language* A TPiet-QL query has the following syntax:

SELECT GIS [ [DISTINCT] SNAPSHOT] list_of_attributes
FROM [OVERLAP] T1 t1,...,Tn tn
WHERE $\Phi$

T1 through Tn represent thematic layers, t1 through tn range over the spatial or *spatiotemporal objects* in these layers. The `OVERLAP` keyword in the `FROM`

clause states that the overlap join semantics must be applied. The list of attributes in the `SELECT` clause defines the schema of the result: a subset of the union of the attributes of the spatiotemporal objects mentioned in the `FROM` clause. The `SNAPSHOT` keyword is used to return a non-temporal relation, eliminating the interval/s associated with each tuple in the query result. The condition $\Phi$ is composed of conjunctions and disjunctions of the function and predicates mentioned above, and can also include the Piet-QL predicate `IN` (and the corresponding OLAP sub-query), to provide compatibility with Piet-QL, and to support OLAP in a spatio-temporal SOLAP scenario. This is why we keep the Piet-QL keyword `GIS` in the `SELECT` clause.

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 coincide it their non-temporal attributes and whose temporal intervals are consecutive, `Coalesce`($\mathcal{G}$) constructs a single spatio-temporal object composed of the non-temporal attributes and the temporal union of all the intervals. We illustrate this semantics extending the city-airport example with a layer containing protected areas, described in the table below (on the right we show the distances between cities and protected areas during different time intervals, although this information is actually not recorded, it must be computed):

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

| cityId | areaId | 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 the query "pairs city-areas such that the distance between them is/was less than 100Km". The query returns tuples of the form $\langle a_i, c_j, Interval \rangle$, where $Interval$ is the interval when they where closer than 100Km from each other. The TPiet-QL expression for this query, and the result are depicted next (note that the tuples in the result have been coalesced).

```
SELECT GIS c,ar
FROM OVERLAP pAreas ar, Cities c
WHERE Distance(c.the_geom,ar.the_geom) < 100
```

| cityId | arealId | 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(p1,[1996,1996]) AND COVERS(p,[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]);
```

*Expressive Power* As we already commented, TPiet-QL is based on the formal model and query language (denoted $\mathcal{L}_t$) described in [12]. Most queries expressible in (denoted $\mathcal{L}_t$) are captured by TPiet-QL. For example, Query 1:

$$\begin{aligned} Q = \{p \mid (\exists e_p)(\exists e_{p_1})(\exists a)(\exists prec) \\ (r^t_{L_{pArea}}(p, e_{p_1}, 1996) \wedge r^t_{L_{pArea}}(p, e_p, Now) \wedge \\ area(e_{p_1}) = a \wedge a > 100 \wedge area(e_p) > a) \wedge \\ Weather(p, 2010, prec) \wedge prec > 120\}. \end{aligned}$$

Here, $Weather(p, 2010, prec)$ is represents a fact table, $area$ is a function computing the area of a spatial object, and $r^t_{L_{pArea}}(p, e_p, t)$ are terms representing the protected areas and their geometric extensions across time (using point-based semantics, for clarity), corresponding to the elements in the model of Section 3. $Now$ represents the current time. The constructs of $\mathcal{L}_t$ are present in the TPiet-QL expression for Query 1. The main difference is that instead of using non-temporal functions over the extensions of spatial objects (e.g., $e_p$) like in $\mathcal{L}_t$, TPiet-QL uses temporal functions over spatio-temporal objects (e.g., $p$). It can be shown that queries expressible in $\mathcal{L}_t$ can be expressed in TPiet-QL since there is a translation for each of the terms in one language to the other. We omit the proof for the sake of space.

## 5 Implementation and Case Study

We now show how the notions above have been implemented and applied to the case introduced in Section 1. Figure 4 shows the graphic user interface (GUI) of the application. Through this GUI, users can edit, save, and execute queries and, as well as configure options for displaying the results. The interface allows

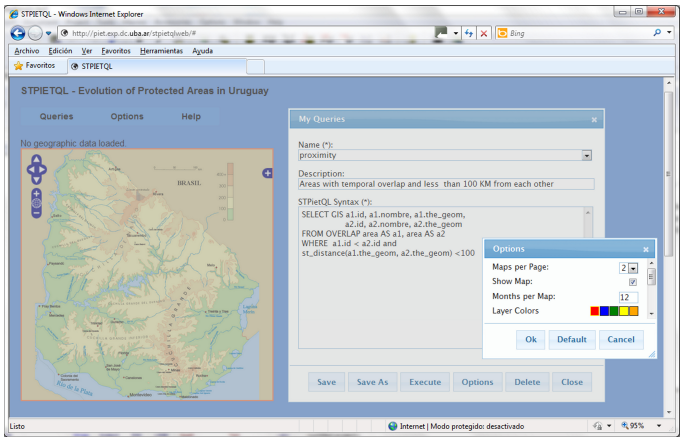


**Fig. 4.** Application Interface

navigating query results through time. To reduce horizontal scrolling, up to three maps can be visualized per page. In Figure 4 we see that the user is editing a query, and configuring a visualization of up to two maps per page. In the background, we see the map of Uruguay, where the protected areas are going to be displayed. Also, to distinguish objects that are in different layers, the user chooses which colors are going to be used for each one, selecting a color pallette.

We show typical queries which practitioners would be interested to pose in this scenario. The first one asks for areas that overlap in time, and are very close to each other, suggesting that they can be eventually extended and merged into a single one, or grouped together for administrative purposes. In addition, the result is filtered using an OLAP query over meteorological data stored in a DW.

**Query 2 (Proximity)** *Pairs of protected areas -expressed in the form (ID, name, geometry)- with temporal overlap and less than 100 KM from each other. Return only pairs such that the distance between both areas and a city affected by temperatures less than 2 degrees Celsius in the last two years (2011 and 2012) is less than 30km.*

```
SELECT GIS a1.id, a1.name, a1.the_geom,
a2.id, a2.name, a2.the_geom
FROM OVERLAP area AS a1, area AS a2, city
WHERE a1.id < a2.id and
  st_distance(a1.the_geom, a2.the_geom) < 100 and
  st_distance(city.the_geom, a1.the_geom) < 30 and
  st_distance(city.the_geom, a2.the_geom) < 30 and
  city.id IN (SELECT CUBE filter([Geography].[City].Members,
  [Measures].[Temperature] < 5)
  FROM [Weather]
  SLICE [Time].[2011],[Time].[2012])
```

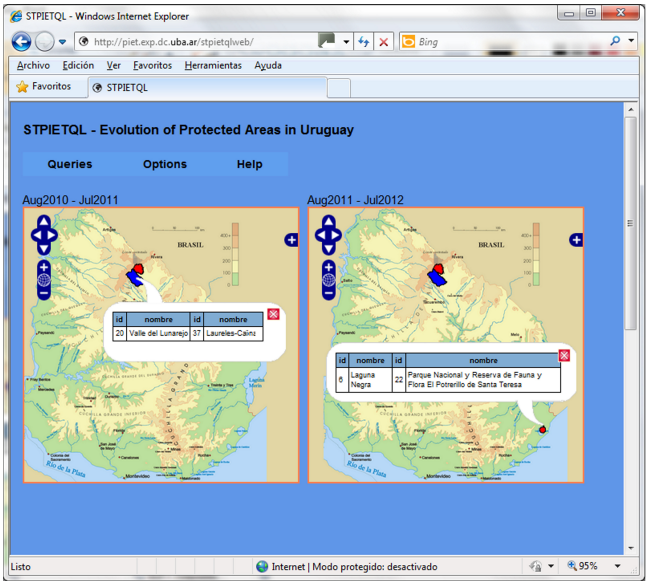


**Fig. 5.** Result of the Proximity query (Query 2).

Figure 5 shows the result. Two pairs of protected areas satisfy the condition: the ones with ids 20 and 37, and the pairs with ids 6 and 22. The latter appears only in the map on the right hand side, because protected area with ID 22 was created on 2010-07-01. Additional information about each tuple in the result is displayed by clicking over it. Note that, given the temporal semantics, the history of all tuples in the result is available and can be navigated.

**Query 3 (Status)** *Protected areas that have changed their status, and that were at a distance of less than 50 km from cities where precipitation in 2010 was greater than 500 mm.*

```
SELECT GIS a1.ID, a1.the_geom, a1.status, a2.status
FROM AREA as a1, AREA as a2
WHERE a1.ID = a2.ID and Before(a1,[a2.from, a2.to] )
  and a1.status<> a2.status and
  st_distance(city.the_geom, a1.the_geom) < 50 and
  cities.ID IN( SELECT CUBE filter([Geography].[City].Members,
  [Measures].[Precipitation] > 500)
  FROM [Weather]
  SLICE [Time].[2010]
```

Figure 6 shows the result. Note that since a `Before` predicate is used in the `WHERE` clause to express changes over time, the overlap join is not used in the

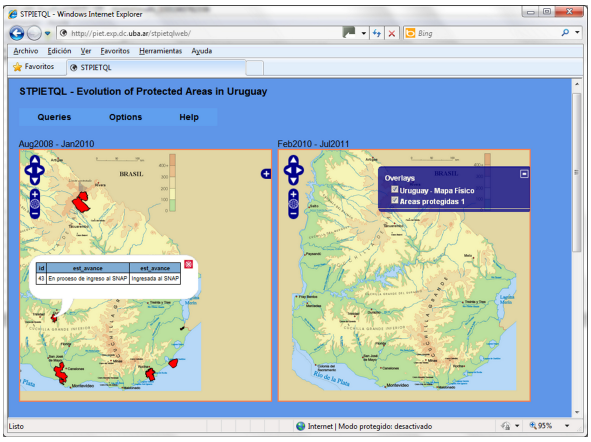


**Fig. 6.** Result for Query 3.

`FROM` clause. The last change of the 'status' property occurred in January of 2010. That is the reason why the map on the right hand side, corresponding to the period between Feb-2010 and Jul-2011 does not display any layer (only the background map). The last example is a snapshot query.

**Query 4 (Snapshot)** *For protected areas whose boundaries have been defined by a decree issued before June 1st, 2010, display ID, name, geometry and status (i.e., omit the temporal attributes).*

```
SELECT GIS DISTINCT SNAPSHOT a1.id, a1.name,
a1.the_geom, a1.status
FROM area AS a1
WHERE a1.boundary = 'decree' and
  STARTSBEFORE(a1, '2010-06-01')
```

Fig. 7 shows the result. All tuples are displayed in the same map, since no temporal information is required due to the use of the `SNAPSHOT` keyword.

## 6 Conclusion and Future Work

We have presented a temporal query language for SOLAP, denoted TPiet-QL, that supports discrete changes in the spatial objects in the thematic layers of a GIS. We also showed how a prototype implementation is applied to a real-world use case concerning the analysis of protected areas in Uruguay. Given the relatively small number of protected areas at this time, the prototype is able to run in reasonable execution times. However, larger amounts of data will require more sophisticated query processing. This constitutes the next step of our work.
**Acknowledgement:** The authors were partially funded by the LACCIR project

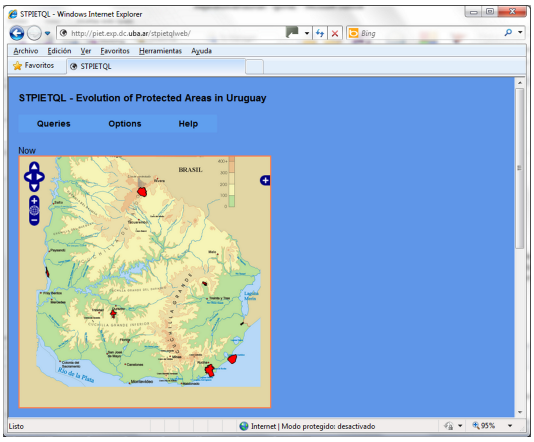


**Fig. 7.** Result for the Snapshot query (Query 4)

"Monitoring Protected Areas using an OLAP-enabled Spatio-temporal GIS". A. Vaisman is partially funded by the "Open Semantic Cloud for Brussels (OSCB)" project, funded by the Brussels Capital Region, Belgium.

## References

1. Tryfona, N., Jensen, C.S.: Conceptual data modeling for spatiotemporal applications. GeoInformatica **3** (1999) 245–268
2. Kimball, R.: The Data Warehouse Toolkit. J.Wiley and Sons, Inc (1996)
3. Gómez, L.I., Haesevoets, S., Kuijpers, B., Vaisman, A.A.: Spatial aggregation: Data model and implementation. Inf. Syst. **34** (2009) 551–576
4. Rivest, S., Bédard, Y., Marchand, P.: Towards better support for spatial decision making: Defining the characteristics of spatial on-line analytical processing (SOLAP). Geomatica **55** (2001) 539–555
5. Tryfona, N., Hadzilacos, T.: Logical data modelling of spatio temporal applications: Definitions and a model. In: IDEAS. (1998) 14–23
6. Tansel, A., Clifford, J., Gadia (eds.), S.: Temporal Databases: Theory, Design and Implementation. Benjamin/Cummings (1993)
7. Hurtado, C.A., Mendelzon, A.O.: OLAP dimension constraints. In: PODS. (2002) 169–179
8. Eder, J., Koncilia, C., Morzy, T.: The COMET metamodel for temporal data warehouses. In: CAiSE. (2002) 83–99
9. Mendelzon, A.O., Vaisman, A.A.: Temporal queries in OLAP. In: VLDB. (2000) 242–253
10. Allen, J.: Maintaining knowledge about temporal intervals. Communications of the ACM 26(11) (1983) 832–843
11. Egenhofer, M.J.: Spatial SQL: A query and presentation language. IEEE Trans. Knowl. Data Eng. **6** (1994) 86–95
12. Gómez, L.I., Kuijpers, B., Vaisman, A.A.: A data model and query language for spatio-temporal decision support. Geoinformatica 15(3) (2011) 455–496