A data model and query language for spatio-temporal decision support

Leticia Gómez · Bart Kuijpers · Alejandro Vaisman

Abstract In recent years, applications aimed at exploring and analyzing spatial data have emerged, powered by the increasing need of software that integrates Geographic Information Systems (GIS) and On-Line Analytical Processing (OLAP). These applications have been called SOLAP (Spatial OLAP). In previous work, the authors have introduced Piet, a system based on a formal data model that integrates in a single framework GIS, OLAP (On-Line Analytical Processing), and Moving Object data. Real-world problems are inherently spatio-temporal. Thus, in this paper we present a data model that extends Piet, allowing tracking the history of spatial data in the GIS layers. We present a formal study of the two typical ways of introducing time into Piet: timestamping the thematic layers in the GIS, and timestamping the spatial objects in each layer. We denote these strategies *snapshot-based* and *timestamp-based* representations, respectively, following well-known terminology borrowed from temporal databases. We present and discuss the formal model for both alternatives. Based on the timestamp-based representation, we introduce a formal First-Order spatio-temporal query language, which we denote $\mathcal{L}_t$, able to express spatio-temporal queries over GIS, OLAP, and trajectory data. Finally, we discuss implementation issues, the update operators that must be supported by the model, and sketch a temporal extension to Piet-QL, the SQL-like query language that supports Piet. **Keywords** OLAP · Spatio-temporal databases · GIS · SOLAP

L. Gómez
Instituto Tecnológico de Buenos Aires,
Av. Madero 399, Buenos Aires, Argentina
e-mail: lgomez@itba.edu.ar

B. Kuijpers
Hasselt University and Transnational University
of Limburg, Gebouw D, 3590, Diepenbeek,
Belgium
e-mail: bart.kuijpers@uhasselt.be

A. Vaisman
Universidad de Buenos Aires, Ciudad
Universitaria,
Pabellon I, Buenos Aires, 1428, Argentina
e-mail: avaisman@dc.uba.ar

1 Introduction

Geographic Information Systems (GIS) have been extensively used in various application domains, ranging from economical, ecological and demographic analysis, to city and route planning [52, 69]. Spatial information in a GIS is typically stored in different so-called *thematic layers* (also called *themes*). Information in themes can be stored in different data structures according to different data models, the most usual ones being the *raster model* and the *vector model*. In a thematic layer, spatial data are typically annotated with classical relational attribute information of (in general) numeric or string type. While spatial data are stored in suitable data structures, associated attributes are usually stored in conventional relational databases. Spatial data in the different thematic layers of a GIS can be mapped univocally to each other using a common frame of reference, like a coordinate system. These layers can be overlayed to obtain an integrated spatial view.

On the other hand, OLAP (On-Line Analytical Processing) [29, 30] provides a set of tools and algorithms that allow efficiently querying multidimensional data repositories usually called Data Warehouses, containing large amounts of data. In OLAP, data are organized as a set of *dimensions* and *fact tables*. In this multidimensional model, data can be perceived as a *data cube*, where each cell contains a measure or set of (probably aggregated) measures of interest. OLAP dimensions are further organized in hierarchies that favor the data aggregation process [10].

1.1 GIS-OLAP decision support

Different data models have been proposed for representing objects in a GIS. The Reference Model proposed by the Open Geospatial Consortium[1] supports the idea of the *Coverage data model*. The *Coverage* data model, introduced by ESRI,[2] and extensively studied by the GIS community, binds spatial objects to non-spatial attributes that describe them. This model was later extended with object-oriented support, in a way that behavior can be defined for geographic features [70]. In spite of the model of choice, there is always the underlying idea of associating spatial objects with objects or attributes stored in object-relational databases.

Nowadays, organizations need sophisticated GIS-based Decision Support System (DSS) to analyze their data with respect to geographic information represented not only as attribute data, but also in maps, probably in different thematic layers. In this sense, OLAP and GIS vendors are increasingly integrating their products.[3]

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

[2] http://www.esri.com

[3] See Microstrategy and MapInfo integration in http://www.microstrategy.com/, http://www.mapinfo.com/solutions/capabilities/business-intelligence.

In this scenario, aggregate queries are central to DSSs. Thus, classical aggregate queries (like "Total sales of cars in California"), and aggregation combined with complex queries involving spatial elements ("Total sales in all villages crossed by the Mississippi river within a radius of 100 km around New Orleans") must be efficiently supported. Moreover, navigation of the results using typical OLAP operations like roll-up or drill-down is also required. These operations are not supported by commercial GIS in a straightforward way. The main reason for this is that the GIS data models discussed above were developed with 'transactional' queries in mind. Thus, the databases storing non-spatial attributes or objects are designed to support those (non-aggregate) kinds of queries. Decision support systems need a different data model, where non-spatial data, consolidated from different sectors in an organization are stored in a data warehouse. For instance, if we were interested in the sales of certain products in stores in a given region, we may consider the sales amounts in a fact table over the three dimensions Store, Time and Product. Dimensions in a data warehouse are usually organized into aggregation hierarchies. For example, stores can aggregate over cities which in turn can aggregate into regions and countries. Each of these aggregation levels can also hold descriptive attributes like city population, the area of a region, etc. Another reason for the lack of OLAP support in commercial GIS is that system integration is not an easy task, and requires a formal data model behind. To fulfill the requirements for integrating GIS and DSS, warehouse data must be linked to geographic data. For instance, a polygon representing a region must be associated with the region identifier in the warehouse. This approach also allows to integrate warehouse data with pre-existing GIS data (or viceversa), designed and implemented independently from each other (we denote this approach as *loosely coupled*). The problem of integrating OLAP-centric systems and GIS-centric systems, has been called SOLAP [6, 7]. However, not many proposals have formally addressed the problem of modeling and querying a GIS-OLAP scenario. One of such works is the Piet system [16, 19].[4] In Piet, spatial, spatio-temporal, and non-spatial data (stored in a data warehouse and/or in the GIS) are integrated in a single framework. To the best of our knowledge, Piet is the only proposal that provides a formal data model and query language for such integration. Moreover, recently, an SQL-like query language for Piet, denoted Piet-QL, has been presented [18].

Tryfona et al. [62, 63] classify spatio-temporal applications according with the kind of support of the changes occurring in the 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*, for instance, a storm, or a stain in a river. That means, spatial objects in thematic layers can be added, removed, split, merged, or their shape may change. In [17, 32] the authors show that the Piet framework supports continuous motion, following the classification above. However, Piet works under the assumption that all objects in a layer remain unchanged across time, i.e., *neither does Piet support objects with discrete changes nor objects combining continuous motion and changing shapes.* For example, Piet supports queries like "Total number of objects going from a three-star hotel to a cheap restaurant on Monday

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

mornings in Antwerp", but not queries like "Number of airports in England in 1998". Addressing discrete changes, continuous motion, and changing shapes, requires a *temporal* data model and query language. We show below that only few proposals address the problem of time-varying geometric objects, and, to the best of our knowledge, none of them is oriented to GIS-OLAP integration. For example, Hermes, a system introduced by Pelekis et al. [46, 48], supports changing objects, although it lacks GIS-OLAP integration capabilities.

In the present paper we extend Piet to support the three types of spatio-temporal data defined in [62] in addition to provide a GIS-OLAP integration framework. This extension positions Piet in the ST-OLAP class, according to the classification given in [65] (see Section 2).

1.2 Two motivating examples

Let us assume the following GIS-OLAP scenario about land property information. In Fig. 1 we show four parcels of land, P1 through P4. They could be characterized, for example, by attributes like type of soil, depth of underground water availability, and vegetation. The parcels are subject to change in shape, area, and in the value of their attributes, and we consider them as composing a single layer denoted L_{land}. Other layers may contain, for instance, airports represented as points, in a layer denoted L_a. We assume, without loss of generality, that each layer contains *only one kind of geometry*, meaning that for representing rivers (as polylines), we must use a different layer. To complete the picture, there is a layer containing rivers (L_r).

There is also non-spatial information stored in a conventional data warehouse. In this data warehouse, dimension tables contain information about stores, schools, and other demographic data. In particular, a dimension *Land* stores information related to the parcels on the left hand side of the figure. The bottom level of this dimension contains the parcel identifiers (p1 through p4). These id's aggregate over a level denoted *region*, which, in turn, aggregates over a level *province*. There is a mapping

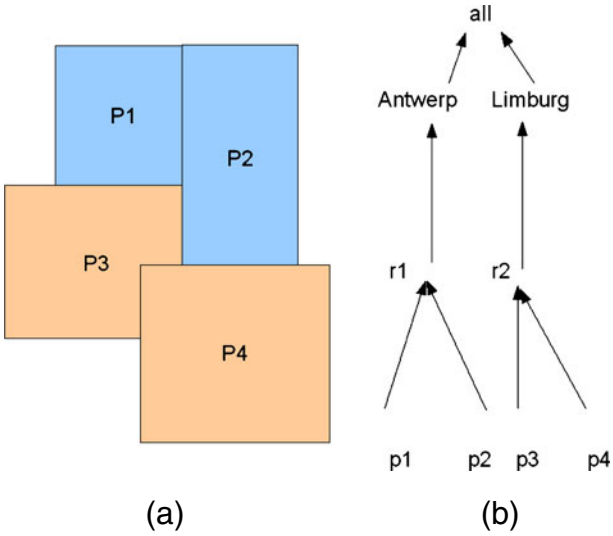

Fig. 1 Motivating example: initial situation. **a** Land partition; **b** land dimension hierarchy

(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 Pi object in the left hand side to the corresponding dimension level member pi in the right hand side).

The scenario above could be represented in Piet in a straightforward way. We remark that integration of GIS and warehouse data could also be possible with existing technologies, through ad-hoc solutions, which would require data exchange between GIS and OLAP applications to be performed. For example, the output of a GIS query must be exported as a dimension of a data cube, and merged for further analysis. However, these solutions, besides requiring many lines of complex coding, are hardly portable. On the contrary, using a system that naturally integrates GIS and OLAP information, a user would only need to bind geographic elements in the maps to the data cube(s) that represent aggregated organizational information, and the system will be ready to work.

Time management requirements appear when, at a certain moment, John, the owner of P3, acquires P4. Then, parcels P3 and P4 must be merged. The new situation is depicted in Fig. 2 (changes have also been performed at the data warehouse, as we can see on the right hand side of the figure). Other changes may occur afterwards, for example, P2 may grow if John decides to sell part of P3-4 in Fig. 2 to the owner of P2. We are now in a discrete changes scenario, where, for instance, we may want to know the history of P3-4, or the production of each existing parcel as of the year 2006, or more complex queries like "Total production by year per square mile for each parcel of land, for the parcels in Antwerp" (note that non-spatial information resides in the data warehouse).

The next example presents a different kind of problem involving continuous motion plus changes in spatial objects, in terms of the classification proposed in [62]. Many industrial processes generate polluting clouds as a side-effect. These clouds are, of course, dynamic: according to the weather conditions, they can move forward

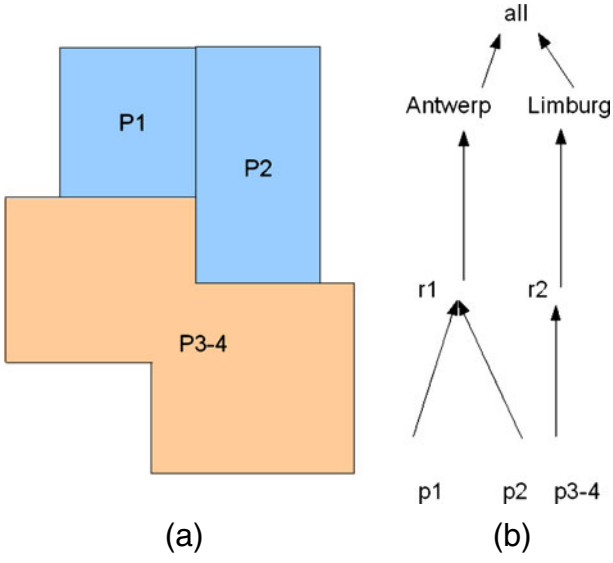

Fig. 2 Motivating example: situation after merging P3 and P4. **a** Changes in spatial objects; **b** changes in the dimension hierarchy

or backward, grow and/or shrink (or even disappear), split into two or more, or merge with other ones. Environmental control agencies monitor these changes in order to keep the situation within certain limits. These limits, of course, depend on the countries involved. Figure 3 shows a typical situation: a polluting cloud denoted C_1 is moving forward (in this case, to the right hand side of the figure). The cloud has first increased its size and changed its shape; it later split into two smaller clouds. Each one of the new clouds further split into two new ones. Finally, C_5 and C_6 merged again into a cloud we denote C_7. This evolution occurs in a certain geographic space (represented in thematic layers, not shown in the figure). Interesting information can be obtained in this scenario, as long as we have a data model and query language allowing representing a dynamic setting. For instance, the evolution of the cloud across time, how far from a school it had passed, how many people were exposed to the effect of the polluting cloud, or how does the total polluting load vary over time, are some of the queries a user may pose. These queries may also involve dimensional information in the warehouse. This evolution is generally recorded through snapshots taken at regular intervals. Figure 4 depicts two such snapshots.

An even more involved situation could be addressed if we consider that, concurrently with the evolution of these polluting clouds, trajectories of moving objects (like cars, or pedestrians) are registered, typically through electronic devices. A user could be interested in asking, for instance, how many persons moved below a polluting cloud for more than one hour without interruption. In this case the answers could only be approximate (through the use of linear interpolation, for instance), since we assume that the information is given in the form of snapshots.

In summary, addressing situations like the ones we have just described requires extending non-temporal SOLAP data models and query languages with temporal capabilities. The reader may ask herself why we need yet another data model and

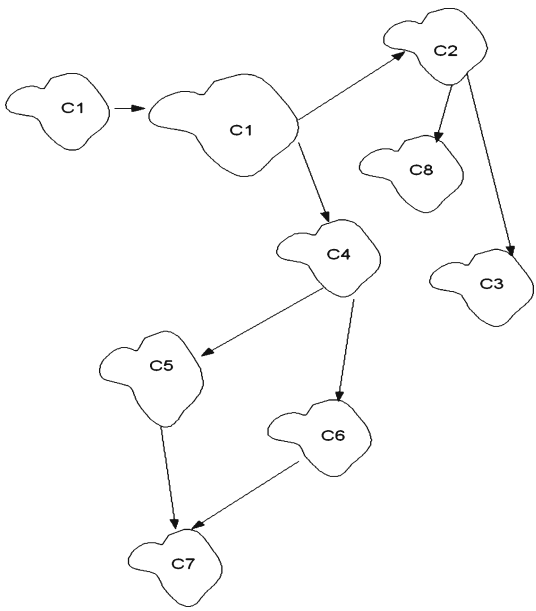

Fig. 3 A pollution cloud changing shape

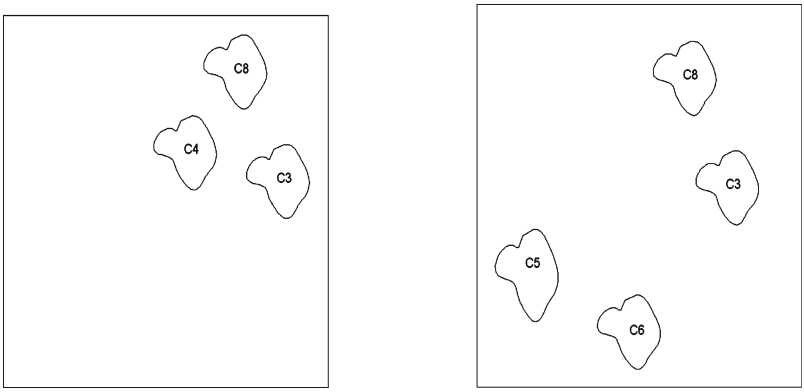

Fig. 4 Snapshot at instant $t = t_1$ (*left*); Snapshot at instant $t = t_2$ (*right*)

system, instead of using existing ones. We argue that this could not be possible because, among other reasons:

- It is likely that in the scenarios we presented above, data warehouses and GIS maps will be built and maintained in an autonomous way (i.e., independently from each other). That is:
  - The GIS existed before the warehouse information was available, or viceversa, the warehouse existed before a GIS was implemented. This means that flexible tools are needed to integrate these pre-existing systems (loosely-coupled approach). Existing SOLAP models and tools are based on the assumption that the system is built as a whole (a tightly-coupled approach).
  - Changes may occur in one or more layers in the map while the data warehouse remains unchanged;
  - The problem may appear the other way around: the warehouse dimensions may change, and the GIS layers remain the same.

  To handle this problem, we need to provide an updatable mapping procedure that binds members in the warehouse dimensions to geometric objects in the GIS layers.
- Even if updates in the warehouse and the GIS were synchronized, not many implementations of temporal warehouses exist [36, 64]. Moreover, it is a well-known fact in OLAP that standard applications have limited updating capabilities.

In Section 7 we discuss the issues above in more detail. In this paper we focus on updates and temporal issues on the GIS side.

1.3 Contributions and paper organization

In this paper we extend the Piet data model with temporal capabilities. The new model supports objects that change across time while maintaining the GIS-OLAP integration paradigm. We first present a brief, although comprehensive overview of the efforts regarding GIS, OLAP and Moving Object data integration (Section 2).

Following a brief overview of the Piet data model (Section 3), we formally define a simplified (also, non-temporal) version of such model, closer to the actual system implementation, that also makes the presentation clearer. In addition, we discuss the implications of these changes over the query language (Section 4). We then study two ways of introducing time in the data model, following standard temporal database literature: snapshot-based representation and timestamp-based representation. In the former, snapshots of the content of the GIS layers are recorded and (most likely, though not mandatory) regular intervals. In the latter, objects in the layers are timestamped with their validity intervals. We present the formal definitions of the temporal model in both cases, in order to support the choice for the timestamp-based approach (Section 5). We then formally present a first-order query language supporting the model and give the intuition of its expressive power through a set of comprehensive examples (Section 6). Finally, we discuss implementation issues. This involves the data structure to support the temporal extension, and how this structure supports the update operators required to give functionality to the model. Also, a temporal extension to Piet-QL is presented, although a formal definition of the syntax and semantics of this language is beyond the scope of this paper (Section 7). We conclude in Section 8.

2 Related work

2.1 Spatial data warehousing and SOLAP

Although some authors have pointed out the benefits of combining GIS and OLAP, not much work has been done in this field. Vega López et al. [34] present a comprehensive survey on spatio-temporal aggregation that includes a section on spatial aggregation. Also, Bédard et al. [7] present a review of the efforts for integrating OLAP and GIS.

Stefanovic et al. [58] and Bédard et al. [6], classify spatial dimension hierarchies according to their spatial references in: (a) non-geometric; (b) geometric to non-geometric; and (c) fully geometric. Malinowski and Zimányi [35] later extend this classification to consider that even in the absence of several related spatial levels, a dimension can be considered spatial. Pourabbas [50] introduces a conceptual model that uses binding attributes to bridge the gap between spatial databases and a data cube. This approach relies on the assumption that all the cells in the cube contain a value, which is not the usual case in practice, as the author expresses. Also, the model requires modifying the structure of the spatial data, and no implementation is reported.

Rivest et al. [53] introduced the concept of SOLAP (standing for Spatial OLAP), a paradigm aimed at being able to explore 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. Although they do not present a formal model for this, the SOLAP concepts and operators have been recently implemented in a commercial tool called JMAP.[5] The authors propose

[5]JMAP was developed by the Centre for Research in Geomatics and KHEOPS, http://www.kheops-tech.com/en/jmap/solap.jsp.

three types of spatial measures: (a) A geometrical shape or set of shapes obtained by the combination of multiple geometric spatial dimensions, for example, via a spatial merge or spatial intersection. (b) A spatial measure resulting from the computation of spatial metric or topological operators. The results of this computation are stored in the data cube cell. Examples of this type of spatial measure are 'area' and 'distance'. (c) A set of pointers stored in the cube cells, to the geometric shapes stored in another structure or software. As the authors point out, current technologies only allow this type of spatial measure. In addition, the authors define a set of topological and metric operators allowing selecting database subsets based on topological and/or geometrical constraints, and spatial roll-up and drill-down operations.

Shekhar et al. [56] presented Map Cube, an operator that, given a so-called base map, cartographic preferences and an aggregation hierarchy, produces an album of maps that can be navigated via roll-up and drill-down operations. The *map cube operator* extends the notion of data cube operator introduced by Gray et al. [20]. The latter generates all possible (2^n) so-called cuboids that can be obtained from a base fact table with n dimensions. The extension basically consists in adding to each element in the cuboid, the geometric extent associated with each group. Thus, the queries corresponding to each cuboid are of the form

```
SELECT A,B,..,AGG(C), AGG_g(Geom)
FROM Base_Fact_Table
GROUP BY A,B,..;
```

Here, A and B are non-spatial dimensions, C is a numeric measure, Geom represents a spatial dimension, AGG is an aggregation function (e.g., SUM), and AGG_g is a geometric aggregation function, for instance the geometric union. In summary, the proposal does not really define a language in the sense of, for example Piet-QL (see below), but defines an operator that can be expressed as a collection of SQL queries. This operator can be used (in conjunction with view materialization techniques) by a visualization tool in order to efficiently roll-up and drill-down along an aggregation hierarchy.

The Spatio Temporal Relational data Model (STRM), introduced by Tryfona and Hadzilacos [61], 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-established models. In this model, like in Piet (which we detail in Section 3), a *layer* is a set of geometric figures like points, lines, regions or combinations of them (although in Piet only one kind of figure is allowed in a layer) with associated values. If a layer represents derived information, it is called *virtual*. The authors define a layer algebra based of four operations over layers, the main ones being the layer overlay and reclassification. The latter is relevant to SOLAP, since it computes the geometric union of adjacent figures if their ranges (i.e., the associated non-geometric attributes) are identical. In other words, the reclassification operation provides the semantics for the cuboids discussed above [56].

Vaisman and Zimányi [65] recently proposed a comprehensive and formal classification for spatio-temporal data warehousing, that goes beyond the informal notion of SOLAP. Using an extended relational calculus supporting aggregation, based on the calculus first introduced by Klug [31] and the abstract data types of Güting et al. [21], define a taxonomy of queries. For example, the SOLAP class of queries is defined as the class containing the queries that can be expressed in

Klug's calculus extended with the spatial data types defined in [21]. Analogously, the ST-OLAP class of queries is the class containing the queries that can be expressed in the calculus extended with spatial and moving types defined in [21].

Piet, named after the Dutch painter Piet Mondrian, Mondrian being the OLAP engine used for the implementation, has been introduced in [16] and [19]. Piet is based on a solid formal model, and the integration between GIS and OLAP is materialized through a function that maps elements in the data warehouse to elements in the GIS layers. A formal (based on first-order logic) query language is also part of the proposal. Piet comes equipped with a query language, Piet-QL [18], that supports the operators proposed by the Open Geospatial Consortium for SQL, adding the necessary syntax to integrate OLAP operations through the OLAP standard MDX.[6] Based on the analysis of the needs of GIS-DSS users, Piet-QL is designed to support basically four kinds of queries: (a) GIS queries filtered using spatial conditions, like "Regions of Belgium 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 100Km from Nijvel". Queries of the first two kinds allow GIS and OLAP users to query maps and warehouses in the usual way. Queries of type (c) and (d) characterize integrated GIS-OLAP queries. The filtering is materialized through a predicate denoted **IN**. To give an idea of how a Piet-QL query looks like, let us consider a query of type (c): "Parcels crossed by the Dÿle river, with production sales greater than 5000 units", which reads in Piet-QL:

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

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**. We give more detail in Section 7, although a full study of Piet-QL is beyond the scope of this paper. Note that unlike [56], we support external data cubes. However, we cannot build a spatial data cube in the sense of the work by Shekhar et al. Piet-QL could build the cuboids if the non-spatial information resides in the GIS part of the system, although non-spatial information in the external data warehouse and spatial information in the GIS cannot appear together in the **SELECT**

[6]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.

clause. In other words, the cuboid shown above could only be written in Piet-QL if the numeric information is stored in the GIS part; nevertheless, aggregations like the following are supported:

```
SELECT GIS p.name,sum(d.area)
FROM district d, province p
WHERE inside(d,p) AND d.pop > 10000
GROUP BY p.name
```

Finally, let us briefly comment on *continuous fields*, a topic not covered in the present paper but which we plan to address in future research (see Section 8). In GIS, continuous fields (from now on, *fields*) model phenomena that can be represented as a function of space and time [28]. A field is formally defined as composed of [39]: (a) a domain $\mathcal{D}$ which is a continuous set; (b) a range of values $\mathcal{R}$; and (c) a mapping function f from $\mathcal{D}$ to $\mathcal{R}$. The joint contribution of the GIS and OLAP communities to the study of models involving OLAP and continuous fields has been quite limited. In a first approach to the problem, Shanmugasundaram et al. [55] propose a data cube representation that deals with continuous dimensions not needing a predefined discrete hierarchy. More recently, Ahmed et al. proposed to use interpolation methods to estimate (continuous) values for dimension levels and measures, based on existing sample data values [2, 3]. Vaisman and Zimányi recently proposed a conceptual model addressing GIS-OLAP scenarios that include fields [66].

2.2 Implementation

From an implementation point of view, Han et al. [24] use OLAP techniques for materializing selected spatial objects and introduce a so-called *Spatial Data Cube*. This model only supports aggregation of such spatial objects (i.e., integration with external warehouses is not studied). Pedersen and Tryfona [45] propose to pre-aggregate spatial facts. First, they pre-process these facts, computing their disjoint parts in order to be able to aggregate them later. That means, pre-aggregation works if the spatial properties of the objects are distributive over some aggregate function. Since this proposal does not consider the geometry, queries like "Give me the total population of cities crossed by a river" are not supported. Other works addressing implementation issues for spatial OLAP (based on indexing and pre-aggregation) are [40, 41, 51, 71].

2.3 Spatio-temporal databases

Many proposals study moving object databases. However, the topic of moving shapes (the issue we address in this paper) has received less attention. The Hermes system [38, 46, 48] provides the functionality needed for handling two-dimensional objects that change location, shape and size, through four kinds of data types: (a) static basic; (b) static temporal; (c) static spatial; (d) moving. Data of type (a) are the standard DBMS data types (integer, real, etc.). Data of type (b) are supported through a library denoted TAU-TLL [46]. Temporal data types supported (extending the ODMG data model) are Timepoint, Period, and Temporal Element. The spatial

data types (c) are provided by the Oracle Spatial library. The object type defined in Oracle, and used by Hermes, is called Sdo_Geometry. The Moving data type (d) encapsulates semantics and functionality of different data types: moving point, linestring, circle, rectangle, polygon, and moving collection. Below these types, a class hierarchy is defined. The basic type is called *moving point*, defined as a sequence of different types of simple functions. It is based on the sliced representation proposed by Güting et al. [21]. Here, a temporal development of a moving object is decomposed in slices such that between each slice, a simple function is defined. The idea is to decompose the definition of each moving type into several definitions, one for each function. The composition of these sub-definitions defines a moving type. *Moving point* is the basis for the other types, like moving circle or moving polygon. These data types are discussed in detail by Pelekis et al. [48]. Objects are provided with a set of operations: (a) topological and distance predicates, like within_distance; (b) temporal functions, like add_unit (adds a new unit of movement), and at_instant (returns the union of the projection of a moving object at a given time instant); (c) distance and direction operators (e.g., the distance between two moving objects); (d) set relationships (e.g., intersection). Also numeric operations on objects are supported, like area or length. In consequence, it would be easy to compute, for instance, the area of an object at a given time instant.

SECONDO [22] is another system supporting the model of Güting et al. [21]. SECONDO is an extensible DBMS platform for building research prototypes, consisting in 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. In spite of their ability to handle spatio-temporal data, neither SECONDO, nor Hermes, are oriented toward addressing the problem of integrating GIS, OLAP and Moving Object data.

Remark 1 At the end of Section 5 we complete the discussion above, providing an analysis of the spatio-temporal data models based on snapshots and timestamping. In Section 7 we also comment on spatial integrity constraints.

In summary, although the proposals above address particular problems, as far as we are aware of, Piet is the first work to address the problem as a whole, from a formal study of the problem of integrating spatial and warehousing information in a single framework that allows obtaining the full potential of this integration, to the implementation of the proposal. Given these features, we believe that Piet is an appropriate starting point for extending SOLAP with temporal capabilities. In the next section we give a brief, yet comprehensive, overview of Piet.

3 Piet data model overview

In this section we provide a short description of Piet, only aimed at making the paper self-contained. The interested reader is referred to [16, 19] for a detailed study. In the Piet conceptual data model, spatial and non-spatial data are integrated in a single framework. The model introduces the concept of *GIS dimension*. As usual in

databases, a GIS dimension has a schema and associated instances. A GIS dimension schema is a collection of graphs such that for each layer there is a graph where (a) there is a node for each kind of geometry, call it G_i; (b) there is an edge between two nodes G_i and G_j if G_j is composed by geometries of type G_i (i.e., the granularity of G_j is coarser than that of G_i); (c) there is a distinguished member *All* that has no outgoing edges; (d) there is exactly one node representing the geometry *point* with no incoming edges. Figure 5 shows the schema of a GIS dimension. At the bottom level of each hierarchy there is the *Algebraic part*. Its associated instances contain the infinite points in a layer, and could be described by means of linear algebraic equalities and inequalities [44]. Above this part there is the *Geometric part*, whose instances contain the identifiers of the geometric elements (i.e., the spatial objects) in the GIS layers and is used to solve the geometric part of a query (e.g., intersections between rivers and parcels). Each point in the Algebraic part may correspond to one or more elements in the Geometric part (e.g., if two or more polylines intersect each other, the same point—in the Algebraic part—is associated with two different polylines in the Geometric part). At the *GIS dimension instance* level, two kinds of rollup *relations* are defined:

- r_{alg}, a 5-ary relation representing the rollup between the algebraic and geometric parts. For instance, $r_{alg}(L_{land}, \text{Pg}, x, y, pg_1)$ says that in the layer L_{land}, a point (x, y) corresponds to a polygon identified by pg_1 in the Geometric part.
- r, a rollup relation representing the rollup between any two levels in the geometric part. For example $r(L_{land}, \text{Li}, \text{Pl}, l_{id}, pl_{id})$ means that in layer L_{land}, a line identified by l_{id} corresponds to a polyline identified by pl_{id}.

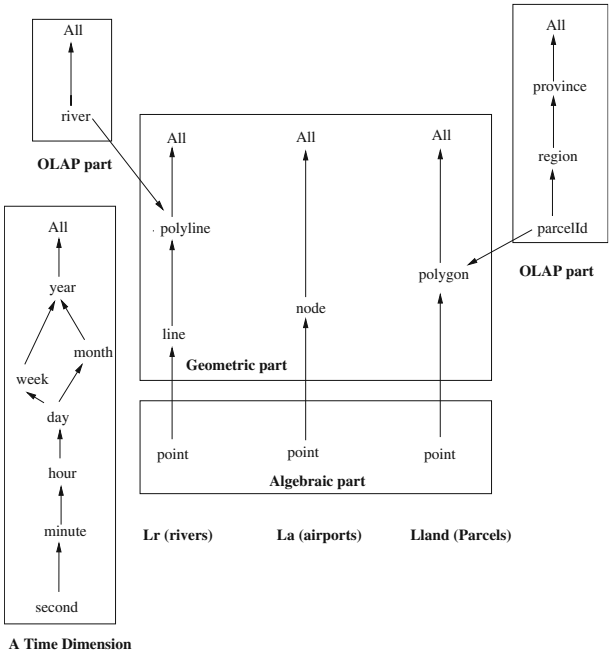

Fig. 5 An example of a *GIS* dimension schema

Finally, there is the *OLAP part* for storing non-spatial data. This part contains the conventional OLAP structures, as defined by Hurtado and Mendelzon [26, 27]. Dimension levels in the OLAP are associated with geometric objects in maps via a function, denoted $\alpha(L, D, \text{dimLevel}, \text{m})$, where L is a layer in the GIS part, D is a dimension in the OLAP part, dimLevel is a level in D, and m is a member in dimLevel. The range of α is the set of identifiers of the spatial objects in L. For instance, $\alpha(L_r, \text{Rivers}, \text{riverId}, r)$ maps an identifier of a river r in the OLAP part (in the dimension level riverId, of the dimension Rivers) to the identifier of the spatial object (a polyline in L_r) in the Geometric part associated with r. The following example is adapted from [19].

Example 1 In Fig. 5, the level *parcelId* in dimension *Land* is associated with a polygon geometry in layer L_{land}. Note that $parcelId \rightarrow region$, and $region \rightarrow province$ in this example ("$A \rightarrow B$" means that there is a functional dependency from level A to level B in the OLAP part [10]). Each dimension level may have attributes associated, like population or number of schools. There is also an OLAP hierarchy associated with the layer L_r at the level of polyline. Notice that since dimension levels are associated with geometries, it is straightforward to associate facts stored in a data warehouse in the OLAP part, in order to aggregate these facts along geometric dimensions.

The association between dimension levels in the OLAP part and levels in the geometric part, i.e., the α function, is actually implemented as a relational table, and any matching algorithm can be used to automatically perform the mapping. This is described in Section 7. We have developed an algorithm based on weighted bipartite graphs matching to perform this mapping in a semi-automatic way.

Figure 6 shows a portion of a GIS dimension instance for the layer L_r in the dimension schema of Fig. 5. We can see that an instance of a GIS dimension in the OLAP part (corresponding to the Schelde river, in Antwerp) is associated via the α function, to the polyline pl_1. For clarity, we only show four different points at the *point* level $\{(x_1, y_1), \ldots, (x_4, y_4)\}$. There is a relation $r_{alg}(L_r, \text{line}, x, y, l_{id})$ containing the association of (x, y) points to the corresponding objects in the *line* level. Analogously, there is a relation $r(L_r, \text{line}, \text{polyline}, l_{id}, pl_{id})$ between the line and

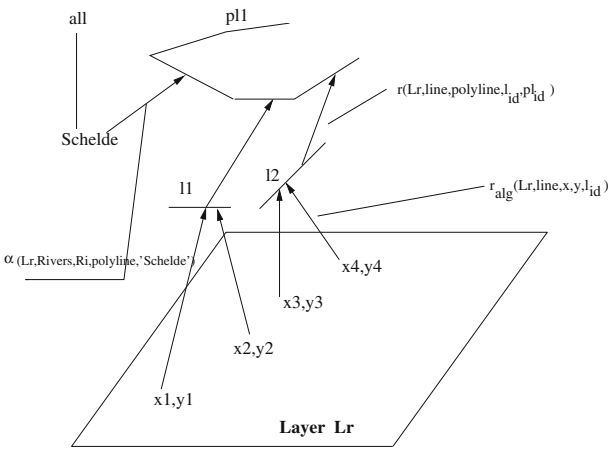

Fig. 6 A *GIS* dimension instance for Fig. 5

polyline levels, in the same layer. Note that the relation representing the association between the algebraic and the geometric parts only has the target level as argument (*line* in this case). For the relation in the geometric part, both, source and target levels are required as attributes (in this example, line and polyline).

Spatial objects in the Geometric part can be associated with *facts*, each fact being quantified by one or more *measures*, not necessarily a numeric value. Consider for instance, a fact table with schema ($polyId$, L_{land}, $grossProduct$), where $grossProduct$ is the measure. Other OLAP fact tables can be defined, also including standard OLAP dimensions. For example, the information above can also be stored in a fact table with schema ($parcelId$, $grossProduct$), where the dimension parcelId contains parcel identifiers in the OLAP part.

3.1 Query language

Based on the data model described above, the notion of *Spatial aggregation* is defined. Spatial aggregation queries are hard to evaluate, since they require the computation of a double integral representing the area where some condition is satisfied. For example, the query "Total gross product of the parcels crossed by the 'Schelde' river" would read:

$$Q \equiv \int\int_C ft_{prd}(x, y, L_c)\, dx\, dy,$$

$$\begin{aligned} C = \{(x, y) \in \mathbb{R}^2 \mid\ & (\exists x')(\exists y')(\exists pg_1)(\exists c) \\ & (r_{alg}(L_c, \text{Pl}, x', y', \alpha(L_r, \text{Rivers}, \text{Ri}, \text{Pl}, \text{'Schelde'})) \ \wedge \\ & \alpha(L_c, \text{Land}, \text{parcelId}, \text{Pg}, c) = pg_1 \ \wedge\ r_{alg}(L_c, \text{Pg}, x', y', pg_1) \ \wedge \\ & r_{alg}(L_c, \text{Pg}, x, y, pg_1))\}. \end{aligned}$$

Piet addresses a class of queries denoted *summable*. Summable queries can be rewritten as sums of functions of the spatial objects returned by the condition 'C'. More formally, a query of the form $Q = \int\int_C h(x, y)\, dx\, dy$ ($C \in \mathbb{R}^2$) is *summable* if $C = \bigcup_{g \in G} ext(g)$, where G is a set of spatial objects, and ext(g) means the *geometric extension* of g (i.e., the subset of $\mathbb{R}^2$ that g occupies), and there exists a function h', such that $Q = \sum_{g \in S} h'(g)$, with $h'(g) = \int\int_C h(x, y)\, dx\, dy$.[7] Here $S = \{g_{id} | \varphi(x, y)\}$, where φ is an FO-formula in a multi-sorted logic $\mathcal{L}$ over the reals, geometric objects, and dimension level members (see Section 4). The vocabulary of $\mathcal{L}$ also contains function names, together with the binary functions $+$ and $\times$ on real numbers, the binary predicate $<$ on real numbers and the real constants 0 and 1. Further, also constants for layers and attributes may appear in $\mathcal{L}$.[8] Atomic formulas in $\mathcal{L}$

[7] A more complete definition of summable queries can be found in [19].

[8] For simplicity, we do not quantify over layers, although the language could be extended to support this.

(basically, syntactic elements for the relations and functions described in the data model) are combined with the standard logical operators $\wedge$, $\vee$ and $\neg$, and existential and universal quantifiers over real variables and attribute variables. A detailed description can be found in [19].

Remark 2 If a query is not summable there is no way of getting rid of the task of computing the integral, since φ can define any semi-algebraic set. For example, the query "Total population endangered by a poisonous cloud described by φ, a formula in first-order logic over $(\mathbb{R}, +, \times, <, 0, 1)$)", is not summable. The simplified model we propose in Section 4.1 cannot express a query like the above, but actually, the Piet-QL language neither supports it.

We now provide an example that gives the flavor of the language.

Example 2 Let us go back to the query "Total gross product of the parcels crossed by the 'Schelde' river":

$$Q \equiv \sum_{\mathrm{g_{id}} \in C} ft_{prd}(\mathrm{g_{id}}, L_{land}).$$

$$\begin{aligned} C = \{g_{id} \mid\ & (\exists x)(\exists y)(\exists pl_1)(\exists p) \\ & (\alpha(L_r, \text{Rivers}, \text{riverId}, \text{Pl}, \text{'Schelde'}) = pl_1 \wedge r_{alg}(L_r, \text{Pl}, x, y, pl_1) \wedge \\ & \alpha(L_{land}, \text{Land}, \text{parcelId}, \text{Pg}, p) = g_{id} \ \wedge\ r_{alg}(L_{land}, \text{Pg}, x, y, g_{id}))\}. \end{aligned}$$

The meaning of the query is the following: $\alpha(L_r, \text{Rivers}, \text{riverId}, \text{Pl}, \text{'Schelde'}) = pl_1$ returns, in the variable pl_1, the identifier of the polyline representing the Schelde river in layer L_r. The relation $r_{alg}(L_r, \text{Pl}, x, y, pl_1)$ contains the x, y coordinates that correspond to the polyline identified with pl_1 (in terms of the model of Fig. 5, the x, y points that roll-up to pl_1 in the Geometric part). Further, $r_{alg}(L_{land}, \text{Pg}, x, y, g_{id})$ contains the association between these coordinates and g_{id}, the variable associated with the identifier of the parcel p by the function $\alpha(L_{land}, \text{Land}, \text{parcelId}, \text{Pg}, p)$. Note that the only free (i.e., not quantified) variable is, precisely, g_{id}. Then, the expression C returns the g_{id} values that make the formula for C true for any instantiation of the quantified variables with values in the database (i.e., we work with the notion of *active domain*). The sum of ft_{prd} (which represents the gross product associated with a polygon g_{id}) over these objects, is then performed.

Considering the classification proposed in [47], *in addition to aggregation*, classic attribute, point, range, distance-based, nearest neighbor and topological queries are supported by the language (in some cases, provided that we add some additional functions). A formal language allows studying a language's properties, but is not, in general, user-friendly. Therefore, based on this language, Piet-QL, the SQL-like query language described in Section 2 was introduced [18].

4 Problem statement and preliminaries

In the remainder of this paper we address the problem of adding temporal capabilities to Piet. More specifically, we study how the Piet data model and the formal FO-query language can be extended to support temporal semantics. A key difference between our approach and typical GIS-oriented ones, is that the model we present is not based upon the object-relational paradigm, that is, we do not provide functionality and semantics through complex data types: they are embedded in the model and query language. This way, the theoretical framework is general enough to be implemented in many ways. We have shown in previous work [19] that this approach can deliver good results for query aggregation, a key issue in data warehousing.

In what follows we restrict ourselves to the *Geometric part* of the data model introduced in Section 3. We can perform this simplification due to the fact that, in practice, we work with object identifiers. Actually, the Piet-QL language works at this level of abstraction. We also consider, without loss of generality, that the GIS dimension is composed of a set of layers, each of them containing a single geometry (e.g., point, polyline). In this way, the clarity of the presentation is enhanced and the formal language becomes more friendly to the reader, while keeping its essence. Moreover, the expressive power is not affected by this simplification, meaning that all summable queries that can be expressed in the full model and language, can also be expressed in the simplified one. In the next sections we give the formal definition of the new (non-temporal) data model and query language.

4.1 A simplified non-temporal data model

We first formally define the simplified non-temporal data model, (based on [19]), since it implies relevant changes to the original work. We then show how these changes impact on the query language.

We have the following sets: a set of layer names **L**, a set of attribute and dimension level names **A**, a set of OLAP dimension names, called **D**, and a set of geometry names, denoted **G**. There is also a set of function names, called **F**. 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 $dom(\mathrm{G})$, 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 1 (GIS-OLAP Dimension Schema) There is a set of layers $L_1, ..., L_k \in \mathbf{L}$, such that for each L_i there is an associated kind of geometry $\mathrm{G}_i \in \mathbf{G}$ (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 [26, 27], where each dimension $\mathrm{D} \in \mathcal{D}$ is a tuple of the form $\langle dname, \mathrm{A}, \preceq \rangle$, such that $dname \in \mathbf{D}$, $\mathrm{A} \in \mathbf{A}$, is a set of dimension levels, and $\preceq$ is a partial order between levels.

We also define a set $\mathcal{A}$ of *partial* functions *Att* with signature $\mathbf{A} \times \mathbf{D} \rightarrow \mathbf{L}$ mapping attributes in OLAP dimensions to layers (see also Definition 2).

A *GIS-OLAP dimension schema* is the tuple $\mathrm{G}_{sch} = \langle H, \mathcal{A}, \mathcal{D} \rangle$

Example 3 The GIS-OLAP dimension corresponding to Fig. 5 has the schema:

$$H(L_r) = \text{polyline}, H(L_a) = \text{node}, H(L_{land}) = \text{polygon}.$$

Following our running example, in the OLAP part we have dimensions *Rivers*, *Airports*, and *Land*. The *Att* functions are:

$Att_1(\text{parcelId}, \text{Land}) = L_{land}$, $Att_2(\text{riverId}, \text{Rivers}) = L_r$, and $Att_3(\text{airportId}, \text{Airport}) = L_a$. Thus, $\mathcal{A} = \langle Att_1, Att_2, Att_3 \rangle$.

In dimension *Land*, it holds that parcelId $\preceq$ region, and region $\preceq$ province. We omit the complete schema for the dimensions in the set $\mathcal{D}$.

Definition 2 (GIS-OLAP Dimension Instance) Let $\mathrm{G}_{sch} = \langle H, \mathcal{A}, \mathcal{D} \rangle$ be a GIS-OLAP dimension schema. A *GIS-OLAP dimension instance* is a tuple $\langle \mathrm{G}_{sch}, \mathcal{I}, \mathcal{A}_{inst}, \mathcal{D}_{inst} \rangle$, where $\mathcal{I}$ is a set of binary relations r_{L_i}, containing the identifiers of the geometric objects (and their extensions), in each layer L_i, such that $H(L_i) = \mathrm{G}$.

Associated with each function *Att* such that $Att(\mathrm{A}, \mathrm{D}) = \mathrm{L}$, there is a function $\alpha \in \mathcal{A}_{inst}$. The arguments of this function are a dimension name (in the OLAP part) D, a layer L, a dimension level $l \in L$, and a member m belonging to l. Note that the kind of geometry for L is uniquely determined by $H(L)$, meaning that we do not need the kind of geometry as an argument. Intuitively, α maps m to the identifier g_{id} of a spatial object belonging to a geometry G, thus binding a data warehouse instance to an instance of a geometry (i.e., a spatial object) in a given layer.

Finally, for each dimension schema $D \in \mathcal{D}$ there is a dimension instance in $\mathcal{D}_{inst}$, defined as in [26, 27], which is a tuple $\langle D, RUP \rangle$, where RUP is a set of rollup functions that relate elements in the different dimension levels (intuitively, these rollup functions indicate how the attribute values in the OLAP part are aggregated). We omit the details, which are studied in the works in the references.

Note that in Definition 2 there is a relation r_L for each layer L, instead of a generic relation r like in the original model. This decision obeys to practical reasons. The simplified model is closer to an actual implementation that the full model. From this point of view, a normalized set of relations seems more appropriate than a single relation. We further illustrate this in Section 7, where we show the data structure for the implementation.

Example 4 For the instance of Fig. 1, we have:

$$\mathcal{I} = \{r_{L_{land}} = \{\langle P1, ext(P1)\rangle, \langle P2, ext(P2)\rangle, \langle P3, ext(P3)\rangle, \langle P4, ext(P4)\rangle\},$$
$$r_{L_a} = \{\langle A1, ext(A1)\rangle, \langle A2, ext(A2)\rangle\}, r_{L_r} = \{\langle R1, ext(R1)\rangle, ...\}\}.$$

In this example, P1 is the identifier of the spatial object representing a parcel, and 'p1' is the identifier of a parcel in the data warehouse dimension level parcelId.

A curator has determined that both of them represent an abstraction of the same physical object. Associated with Att_1 in Example 3 we have:

$$\{\alpha(L_{land}, \text{Land}, \text{parcelId}, \text{'p1'}) = P1,\ \alpha(L_{land}, \text{Land}, \text{parcelId}, \text{'p2'}) = P2, ...\}.$$

Analogously, there are α functions associated with Att_2 and Att_3.

The data model allows to seamlessly introduce other kinds of external information. For example, *moving object data* can be associated with geographic information, to produce information of interest about the trajectories followed by moving objects registered, for instance, through GPS or GSM devices (following [62] we denote this as continuous motion). Geographic information is stored in the GIS layers, using the data structures described in the present section. Trajectories are given as samples of the form (O_{id}, x, y, t), stating that an object O_{id} is at the coordinate x, y, at an instant t. They are stored in a table denoted MOFT (moving object fact table). These new structures can be easily introduced into the query language described below. We give examples in Sections 6 and 7.

4.2 Query language for the simplified Piet data model

Leaving out the algebraic part of the formal data model implies that we must modify the FO-language $\mathcal{L}$ in two ways: first, the terms corresponding to the rollup relations must be removed; second, the binary relations in $\mathcal{I}$ must be added to the syntax, together with the topological predicates and functions that operate over the extensions of the geometric objects. A substantial change to the original language is the introduction of *variables for the extensions* of the geometric objects. These variables can also be existentially or universally quantified, and they provide a direct link between theory and implementation, since these extensions can be mapped to data of the types supported by standard GIS languages. The introduction of the extensions, topological predicates and functions, allow preserving the expressive power of the language. Using these three features we can define the same sets 'C' as in the original model. We give the intuition of this by means of an example. Consider again the query "Total gross product of the parcels crossed by the 'Schelde' river". In the simplified model, it reads:

$$Q \equiv \sum_{\text{g}_{\text{id}} \in C} ft_{prd}(\text{g}_{\text{id}}, L_{land}).$$

$$\begin{aligned} C = \{g_{id} \mid (\exists pl)(\exists p)(\exists pl)(\exists e_p)(\exists e_g) \\ (\alpha(L_r, Rivers, \text{riverId}, \text{'Schelde'}) = pl \wedge \\ \alpha(L_{land}, \text{Land}, \text{parcelId}, p) = g_{id} \wedge \\ r_{L_{land}}(pl, e_p) \wedge r_{L_r}(g_{id}, e_g) \wedge intersects(e_p, e_g))\}. \end{aligned}$$

The semantics is analogous to the one explained in Example 2. However, two facts should be noted in the expressions above: (a) the extensions of pl and g_{id} (denoted, respectively, e_p and e_g) are of a different kind: one is a polyline (in postGIS terminology, a Linestring type) and the other one is a polygon; (b) the conjunction

$$r_{L_{land}}(pl, e_p) \wedge r_{L_r}(g_{id}, e_g) \wedge intersects(e_p, e_g)$$

replaces the expression:

$$r_{alg}(L_{land}, \text{Pg}, x, y, g_{id}) \wedge \alpha(L_{land}, \text{Land}, \text{parcelId}, \text{Pg}, p) = g_{id} \wedge r_{alg}(L_r, \text{Pl}, x, y, pl_1)\}.$$

that computes the geometries that intersect each other. The set of function names **F** defined above, allows the introduction of these functions in the FO-language. The simplified data model has the same expressiveness of the general model, *provided that the set* **F** *contains the appropriate functions.* That is, the set **F** allows the implementation of a flexible, extendible language.

We are now ready to define the temporal data model. We do this in the next section.

5 Introducing time into Piet

A large number of temporal data models can be found in the relational database literature [59]. Among them, the most popular ones are the *snapshot* and the *timestamp* models. In the former, there is a function from the time domain to the database such that, for each instant t, we have a set of timestamped relations of arity two where the first attribute is t, and the second one is a set of tuples valid at t. In the *timestamp* model, each relation is timestamped at the tuple or attribute level. Based on the above, in this section we study two alternative ways of introducing time into the data model: (a) A *snapshot-based* representation, where a temporal label is attached to the *layers* in the GIS; (b) A *timestamp-based* representation, where the *objects* in the different GIS layers are timestamped with their validity time instants. We give the formal definition for both alternatives and discuss them. First, we need some preliminary definitions and assumptions.

5.1 Representation of time

We consider Time as a new sort (domain) in our model. Toman [60] 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. The point-based representation leads to a clean and elegant syntax and semantics for temporal query languages. In the sequel, we work with point-based temporal domains, although in the implementation, of course, we use interval-based domains (see Section 7).

Definition 3 (cf. [60]—Point-based Temporal Domain) Let T be a set, and $<$ a discrete linear order without endpoints on T. Then, the structure $T_P = (T, <)$ is the *Point-based Temporal Domain*.

The elements in the carrier of T model the individual time instants, and the linear order $<$ models the succession of time. We consider the set T to be **N**, standing for the natural numbers. Unless noted, we work with the former, like in most temporal database applications.

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 [59]. We assume *valid* time support in this paper. Also, as usual in temporal databases, a distinguished variable *Now* represents the (moving) current time instant.

5.2 Model requirements

Our model must manage the types of changes described in [47], namely: changes in attribute, geometry, topology (and all possible combinations of the three). We assume that spatial objects in a layer can be: created, deleted, modified, merged with other ones, and split into several ones, although the formal temporal model is independent of these operations. We discuss these operations in Section 7. Note that analogous operations have been defined for temporal data warehousing [25, 64]. Further, we are aimed at supporting spatio-temporal queries *with or without aggregation.* In this way, we address, among other ones, the following kinds of queries, discussed by Pelekis et al. [47]:

- Spatio-temporal queries about locations. For example: (a) Who is the current owner of parcel p1? (b) Who was its owner in 1997? (c) Parcels crossed by rivers; (d) parcels limiting with P1; (e) total number of airports in P1 in 2006.
- Queries about temporal relationships. For example: (a) average cereal production in parcel P2 the year it was sold to its current owner; (b) Total sales in gas stations the year when airport A1 was opened.
- Queries about moving regions (changing shapes) and trajectories (continuous motion), like "number of people exposed to a polluting cloud in November, 2006".

In other words, we want to keep Piet's expressive power, while adding temporal support. *Also, note that some of the queries above do not even require the presence of a data warehouse,* and are still supported by our proposal.

Remark 3 We do not discuss the temporal aspects of the data warehouse except when necessary for the presentation. A study of this problem can be found, among others, in [14, 36, 37]. Besides, supporting spatio-temporal data warehousing requires that each time an update is produced on one side of the system (i.e., the OLAP or GIS parts), the other side must be updated accordingly, as well as the mapping function α. We comment on this in Section 7.2 below, and assume that the update of α is performed as required.

Remark 4 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 (nevertheless, the extension of a geometric object p can change, and its given, at a given instant, by $ext(p)$). 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 values of the geometric features in the layers and the members of the OLAP dimensions can change (i.e., we support changes in the instances) and these changes may also impact the mapping functions. From the point of view of the classification proposed in [65], we can say

that our proposal falls somewhere in between of the ST-OLAP and ST-TOLAP classes (spatio-temporal OLAP and spatio-temporal TOLAP, respectively), since there is no full support of temporal data warehouses.

Now, we present the two alternative data models in a formal way. We first introduce the snapshot-based temporal extension for Piet, and then the timestamp-based one. In both cases we work with the point-based temporal domain T_P. We also define a *time dimension* D_T and denote μ, a level in this dimension, the *granularity* of a GIS temporal schema.

5.3 Snapshot-based representation

In this representation, the mechanism we use for adding the time dimension to GIS layers consists in labeling the layer with intervals. A set of intervals is called a *temporal element*, following standard temporal database notation. First, we need to define the concept of *lifespan*.

Definition 4 (Lifespan) The *lifespan* of a GIS thematic 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 $\mathcal{L}$ in a GIS, denoted $lifespan(\mathcal{L})$, is the union of the lifespans of all the layers in $\mathcal{L}$.

Definition 5 (GIS-OLAP Dimension Schema) A *snapshot-based GIS-OLAP dimension schema* TG_{sch} is the tuple $\langle H, \mathcal{A}, \mathcal{D}, \lambda, \mu \rangle$, where μ is a level in the Time dimension D_T as defined above, λ is a relation in $dom(\mu) \times \mathbf{L}$, where each tuple (t, L_i) indicates that there exists a snapshot of L taken at instant t, and H, $\mathcal{A}$, and $\mathcal{D}$, are the ones of Definition 1, satisfying the following constraints:

- H is constant throughout the lifespan of the GIS.
- For each layer $L \in \mathbf{L}$, the functions Att are defined only in $lifespan(L)$. where $lifespan(L)$ is the collection of all the instants corresponding to the snapshots of L.
- Once a layer is created, the functions $Att \in \mathcal{A}$ do not change. For example, $Att_1(parcelId, Land)$ returns L_{land} throughout the complete lifespan of L_{land}.
- The schema of the dimensions in $\mathcal{D}$ is constant during the lifespan of the GIS.

Remark 5 Theoretically, for each time unit there is a snapshot of the database (the set of layers, in this case). However, in practice, it suffices to assume that a snapshot of the database is taken each time an update occurs. Thus, between two consecutive snapshots we can assume that no changes occurred.

Definition 6 (GIS-OLAP Dimension Instance) Let TG_{sch} be a Temporal GIS-OLAP dimension schema. A *snapshot-based temporal GIS-OLAP dimension instance* is a tuple $\langle \mathrm{TG}_{sch}, \mathcal{I}^t, \mathcal{A}^t_{inst}, \mathcal{D}^t_{inst} \rangle$, where:

- $\mathcal{I}^t$ is a family of sets of relations $r_{L_i,t}$ containing the geometric elements (and their extensions) in L_i at an instant t. Thus, $\mathcal{I}^t$ contains, for each layer L_i, a set of snapshots at instants $t_1, ...t_m$, such that each one of these instants is in $lifespan(L_i)$.

- Associated with each function Att such that $Att(\mathrm{A}, \mathrm{D}) = L$, there is a set of functions $\alpha \in \mathcal{A}^t_{inst}$, containing the function $\alpha[T]$ as of the time instants $T = \{t | t \in (t_1, ...t_n) \wedge t \in lifespan(L)\}$. Thus, $\mathcal{A}^t_{inst}$ is a family of sets of snapshots (one set for each function α).
- $\mathcal{D}^t_{inst}$ is a family of sets of dimension instances, one set for each dimension schema $D \in \mathcal{D}$ in TG_{sch}. Each set contains the snapshots of the dimension instances at instants $t_1, ...t_m$, such that each one of these instants is in $lifespan((L))$.

Example 5 Let us suppose that layer L_{land} has the lifespan $\{0, 5, 10, 20, 40\}$. If the instance of Fig. 1 corresponds to the situation at $t = 5$, and the instance of Fig. 2 corresponds to the current situation, and there was only one airport at $t = 5$, we may have, for example:

$$\mathcal{I}^t = \{r_{L_{land},5} = \{\langle P1, ext(P1)\rangle, \langle P2, ext(P2)\rangle, \langle P3, ext(P3)\rangle, \langle P4, ext(P4)\rangle\}, ...,$$
$$r_{L_{land},40} = \{\langle P1, ext(P1)\rangle, \langle P2, ext(P2)\rangle, \langle P3-4, ext(P3-4)\rangle\},$$
$$r_{L_a,5} = \{\langle A1, ext(A1)\rangle\}, ...\}.$$

Analogously, for the function in $\mathcal{A}^t_{inst}$ associated with Att_1 in Example 3.

$$\{\alpha(L_{land}, \text{Land, parcelId, 'p1'})[5] = P1, \ldots, \alpha(L_{land}, \text{Land, parcelId, 'p3-4'})[Now] = P3\text{-}4, ...\}.$$

5.4 Timestamp-based representation

We now discuss how we can introduce time into the data model by means of *object timestamping*. That is, instead of working with a set of layer snapshots, we associate a time instant to each object in a layer (recall that we work with the *abstract* temporal database concept, although for efficiency reasons, actual implementations usually encode these instants in intervals). Thus, the lifespan of a layer is implicit in the lifespan of the objects composing it. The definitions of temporal schema and instance change accordingly.

Definition 7 (Timestamp-based GIS-OLAP Dimension Schema) A *timestamped temporal GIS-OLAP dimension schema* TG_{sch} is the tuple $\langle H, \mathcal{A}, \mathcal{D}, \mu\rangle$, where μ is a level in the Time dimension as defined above, and H, $\mathcal{A}$, and $\mathcal{D}$ are the ones of Definition 1, satisfying the following conditions:

- A layer is created when the first object is added to it.
- H is constant throughout the lifespan of the GIS.
- For each layer $L \in \mathbf{L}$, the function Att is defined only in $lifespan(L)$, where $lifespan(L)$ is the collection of all the time instants associated with the geometric objects in L.
- The functions $Att \in \mathcal{A}$ do not change with time. This means that, for example, $Att_1(parcelId, Land)$ will always return L_{land}.
- The schema of the dimensions in $\mathcal{D}$ is constant during the lifespan of the GIS.

Note that we do not need the relation λ because the lifespan of a layer is determined by the lifespan of its objects.

Definition 8 (GIS-OLAP Dimension Instance) Let TG_{sch} be a Temporal GIS-OLAP dimension schema. A *timestamped temporal GIS-OLAP dimension instance* is a tuple $\langle \text{TG}_{sch}, \mathcal{I}^t, \mathcal{A}^t_{inst}, \mathcal{D}^t_{inst} \rangle$, where:

- $\mathcal{I}^t$ is a set of sets 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.
- Associated with each function Att such that $Att(\text{A}, \text{D}) = \text{L}$, 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 $A \in \mathbf{A}$, and G is such that $H(L) = G$ in TG_{sch}.
- $\mathcal{D}^t_{inst}$ is a set of sets of dimension instances, one set for each dimension schema $D \in \mathcal{D}$ in TG_{sch}. Each set contains the dimension instances represented in the timestamp model (see [36] for details).

Definitions 7 and 8 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 6 Consider again the layer L_{land}, with lifespan {[0, 5], [10, 20], [40, 80]}, and the situation in Example 5. We instantiate the *current* instant with the value '80' to be consistent with the definition of the time domain T_P. In an actual implementation, the distinguished variable Now is used. Then,

$$\begin{aligned}
\mathcal{I}^t = \{ & r^t_{L_{land}} = \{\langle P1, ext(P1), 0\rangle, \langle P1, ext(P1), 1\rangle, \langle P1, ext(P1), 2\rangle, \langle P1, ext(P1), 3\rangle, \\
& \langle P1, ext(P1), 4\rangle, \langle P1, ext(P1), 5\rangle, \langle P1, ext(P1), 10\rangle, ..., \\
& \langle p3 - 4, ext(p3 - 4), 80\rangle, ...\}, ..., \\
r^t_{L_a} = \{ & \langle A1, ext(A1), 0\rangle, \langle A1, ext(A1), 1\rangle, ..\}, ...\}.
\end{aligned}$$

In a *concrete* temporal database, the former could be encoded in the form:

$$\begin{aligned}
\mathcal{I}^t = \{ & r^t_{L_{land}} = \{\langle P1, ext(P1), \{[0, 5], [10, 20], [40, 80]\}\rangle, ...\}, \\
r^t_{L_a} = \{ & \langle A1, ext(A1), \{[0, 5], [10, 20], [40, Now]\}\rangle, \langle \\
& A2, ext(A2), \{[10, 20], [40, 80]\}\rangle, ...\}\}
\end{aligned}$$

Analogously, for $\mathcal{A}^t_{inst}$,

$$\begin{aligned}
\{\{ & \langle \alpha(L_{land}, \text{Land}, \text{parcelId}, \text{'p1'}, 0) = P1\rangle, \ldots, \\
& \alpha(L_{land}, \text{Land}, \text{parcelId}, \text{'p1'}, 80) = P1\rangle\}, ...\}.
\end{aligned}$$

In a *concrete* temporal database, the former would be encoded in the form:

$$\begin{aligned}
\{[& \langle \alpha(L_{land}, \text{Land}, \text{parcelId}, \text{'p1'}, [0, 5]) = P1\rangle, ..., \\
& \langle \alpha(L_{land}, \text{Land}, \text{parcelId}, \text{'p1'}, [10, 20]) = P1\rangle....]\}.
\end{aligned}$$

The fact tables are defined analogously to the non-temporal case, and we omit details to avoid redundancy.

Technically, the snapshot and timestamp models have the same expressiveness. This means, we can switch between each other and still be able to represent the same situation. This happens if there is a snapshot (or an encoding of a snapshot) for every time instant in T_P. In a real-world situation this cannot be guaranteed, since snapshots are generated, in the best case, at regular time intervals. Therefore, in practice, we cannot guarantee the equivalence between the two models, and we need to choose one of them as the basis of our spatio-temporal data model.

The snapshot model supports essentially transaction time. On the other hand, the timestamp model favors the valid time approach, allowing retroactive updates. Also, for the timestamp model, time is added to the model just as a new data type, whereas in the snapshot-based model this requires different data structures to be added. In this sense, two typical representations exist: the first normal form (1NF) and the non-1NF timestampings. In the first one, a new tuple is generated each time an object changes (this is also called *tuple timestamping*). In the latter, there is always a unique tuple for each object, and attributes are temporally ordered lists containing the value of the attribute in different intervals. This is also called *attribute-timestamping.*

With respect to querying, it is a well-known fact that the snapshot model makes it difficult to evaluate queries of the form "give me all the time instants where a formula φ holds in the database".

Examples of both kinds of models discussed here exist (not for SOLAP, but for spatio-temporal GIS). Space-time composite (STC) is a model based on attribute timestamping proposed by Langran and Chrisman [33]. The model represents the world as a set of spatially homogenous objects in a layer. Every space-time composite has its unique temporal course of changes in attributes, and attribute changes are recorded at discrete times. The model does not capture temporality among attributes across space, and updating the database requires reconstruction of the STC units. Therefore, the database needs to be re-organized when geometric and topological relationships among STC units change. On the other hand, a model based on snapshots has been proposed by Armstrong [5], with the problems already discussed, common to the general snapshot approach.

Other spatio-temporal data models exist, based on the Object-oriented paradigm [67, 68] (some of them were discussed in Section 2) or Event-Oriented data models [49]. However, since Piet is based on the relational model, we do not consider those models in this discussion. Pelekis et al. [47] give a comprehensive overview of spatio-temporal models.

We conclude that in order to extend Piet with spatio-temporal GIS-OLAP capabilities, the timestamp-based model fulfills the requirements stated in Section 5.2 better than the snapshot-based model. Thus, in what follows, we work with the former.

6 Query language

We now study the temporal extension of the language $\mathcal{L}$ introduced in Section 3. We denote this language $\mathcal{L}_t$. Recall that in Section 4 we introduced the temporal domain T_P. To support temporal queries we need to add this sort, and the relations

studied in Section 5, to the FO-language of Section 4.2. We stress the fact that the FO-language is not aimed at being a working query language for Piet, given that no many regular users could be able to write queries in this language, but it is a clean and formal tool for gaining insight on what could be expressed in FO. Further, in databases, formal languages can be the basis of more user-friendly query languages. Finally, note that in this simplified version of the data model, each structure can be almost straightforwardly mapped to a construct in a language like postGIS.

6.1 $\mathcal{L}_t$ syntax and semantics

In the $\mathcal{L}_t$-definable sets considered in the previous section, we can see that there are variables of different kinds, like O_{id}, t, x, y and g_{id} (actually, $\mathcal{L}_t$ is a multi-sorted first-order logic). We now define $\mathcal{L}_t$ formally.

Definition 9 (The $\mathcal{L}_t$ Query Language) The first-order query language $\mathcal{L}_t$ has the following types of variables: *real variables* $x, y, z, \ldots$, ranging over $\mathbb{R}$; *name variables* $O_{id}, \ldots$, ranging over Object identifiers in the database; *geometric identifier variables* $g_{id}, \ldots$, ranging over identifiers of geometric objects in the database; *variables for members of dimension levels*, $a, b, c, \ldots$, (which are also used for dimension level attributes), ranging over members in the dimension levels in the data warehouse; *temporal variables* defined over T_P, denoted $t, \ldots, t_i, \ldots t_n$; and *extension variables*, e_i, e_j, which range over geometric data in the database (e.g., in a sense, these are a kind of second order variables). Atomic formulas in $\mathcal{L}_t$ are built via existential and universal quantification over these variables, and the usual logical connectives $\wedge, \vee, \neg$, plus a set of functions and relations. We now define how these atomic formulas are formed.

Terms The following are the only *terms* in the language.

- A function symbol $f_D^{l_i \to l_j}(t)$, where l_i and l_j are levels in the dimension D, and t is a temporal variable, is a term; There is one function symbol for each rollup function in the OLAP part.
- A function symbol $\beta_D^{A \to B}$ that maps elements of A to elements of B in dimension D (in the OLAP part), is a term; for every dimension level A, and every attribute B of A, denoted $A.B$, there is one function symbol.
- A function symbol $\alpha(L, D, A_i, a, t)$, where A_i is a level in dimension D, L is a layer, a is a variable for a dimension level member, and t is a temporal variable, is a term; There us one function symbol for every α function in A^t_{inst} associating the OLAP and GIS parts.
- Metric functions, e.g., $distance(e_1, e_2)$ where e_1 and e_2 are extension variables, are terms;
- There are functions that can be applied to the alpha-numeric data in the OLAP part (for instance, the function *concat* on string values); these functions are also terms in the language.

Atomic Formulas In $\mathcal{L}_t$ we have arithmetic operations $+$ and $\times$, the constants 0 and 1, and the relation $<$ for real numbers and temporal variables. We also assume the

equality relation for all types of variables. There are numeric, string, and temporal constants. Atomic formulas are formed as follows.

- For every relation r^t_L in $\mathcal{I}^t$, we have a relation symbol $r^t_L(g, e, t)$, where g is a geometric identifier variable for a geometry in L, e is an extension variable, and t is a temporal variable; thus, r^t_L in $\mathcal{I}^t$ is a formula.
- There are topological relations $r_{top}(e_1, e_2)$ (e.g., intersects, adjacent, touches), where e_i are extension variables; thus, $r_{top}(e_1, e_2)$ is a formula.[9]
- For every external fact table of arity n, like, for instance, the MOFT mentioned in Section 4, we have an n-ary relation $\mathcal{M}_i$; $\mathcal{M}_i$ is a formula.
- If v is a variable of one of the sorts defined above, $(\exists v)$ and $(\forall v)$ are formulas.
- If *term*1 and *term*2 are terms, *term*1 = *term*2 is a formula.
- If $\mathcal{F}_1$ and $\mathcal{F}_2$ are formulas, $\mathcal{F}_1 \wedge \mathcal{F}_2$ and $\mathcal{F}_1 \vee \mathcal{F}_2$ are formulas. Also, if $\mathcal{F}$ is a formula, $\neg\mathcal{F}$ is a formula.
- Parentheses are also allowed, with the usual precedence.
- Nothing else is a formula.

The $\mathcal{L}_t$ language could be used in two ways: (a) As a language to define the aggregation region C of Example 2; (b) As a query language in itself. Also in this case, aggregation could be applied. For example, the Count, Max and Min operators could be applied to sets of the form $\{t \mid \phi(t)\}$, when the $\mathcal{L}_t$-definable condition ϕ defines a finite set of time instants. Or the TimeSpan operator could be applied when ϕ defines an infinite, but bounded set of time instants. The semantics of Count, Max and Min is clear and TimeSpan returns the difference between the maximal and minimal moments in the set. Other (temporal and non-temporal) aggregate operators can be applied in analogous way.

Example 7 Let us start with the query "Total production of the parcels crossed by the river 'Schelde' in 2004":

$$Q \equiv \sum_{\mathrm{g_{id}} \in C} ft_{prd}(\mathrm{g_{id}}, L_c, 2004).$$

$$C = \{g_{id} \mid (\exists p)(\exists p_l)(\exists p1)(\exists e_g)(\exists e_r)$$
$$(\alpha(L_r, \text{Rivers, Ri}, p1, 2004) = pl \wedge p_1.name = \text{'Schelde'} \wedge$$
$$\alpha(L_{land}, \text{Land, parcelId}, p, 2004) = g_{id} \wedge$$
$$r^t_{L_{land}}(g_{id}, e_g, 2004) \wedge r^t_{L_r}(pl, e_r, 2004) \wedge intersects(e_g, e_r))\}.$$

Here, we do not use temporal variables. The example is aimed at showing that the model supports these kinds of historical queries. The fact table stores, for each tuple $\langle g_{id}, ext(g_{id}), t\rangle$ in L_c, the production associated with g_{id} at instant t. For example, if $\langle g1, ext(g_1), 2004\rangle$ is a tuple in $\mathcal{I}^t$ for L_c, there is an attribute *population* associated with g_1. Besides, $r^t_{L_{land}}(g_{id}, e_g, 2004)$ and $r^t_{L_r}(pl, e_r, 2004)$ represent the geometric extensions as of 2004 of the spatial objects identified by g_{id} and pl (parcels and rivers, respectively). Thus, if in 2004 pl and g_{id} intersected, the latter contributes to the

[9] Egenhofer and Herring defined the 9-intersection model for binary topological relations [15], where every set of 9-intersections, represented as a 3×3 matrix, describes a binary topological relation.

aggregation. Note that $intersects(e_g, e_r)$ is not explicitly a function of time, because e_g and e_r correspond to the year 2004. Finally, note that the geometric identifiers are unique at a given time instant (in the granularity of the layer).

Remark 6 We do not get into granularity details, which have been extensively studied in the temporal database literature. We assume that the necessary functions for date conversion would be available [8, 13, 57], if needed, for handling different temporal granularities between the warehouse and GIS historical data, or even between different layers in the GIS.

Definition 9 describes the syntax of the language $\mathcal{L}_t$. The interpretation of all variables, functions, relations, and constants is standard, as well as that of the logical connectives and quantifiers. We illustrate the semantics through a series of examples. These examples are also aimed at showing that the four kinds of queries supported by Piet-QL are also supported in our temporal query language: GIS, OLAP, GIS filtered by OLAP cubes (or fact tables), and OLAP queries filtered by GIS constraints (see [18] for details), now including the time dimension.

The following queries refer to our first running example, introduced in Section 1, namely the Land Information System. Recall that we have thematic layers containing information about land use, rivers, and airports, labeled L_{land} and L_r, and L_a, respectively. Historical data recording the gross product for each parcel in L_{land} is stored in a *fact table* ft_{prd}. Unless indicated, we assume granularity *year*. We start with an aggregate query over a fact table defined over attributes in the map layers. This query also includes attributes and distance computation.

Q_1: Total production in parcels within 100 KM from *Brussels*, in 2006

$$Q_1 \equiv \sum_{\mathrm{g_{id}} \in C} ft_{prd}(\mathrm{g_{id}}, L_c, 2006).$$

$$\begin{aligned} C = \{g_{id} \mid\ & (\exists pg_1)(\exists e_g)(\exists e_p)(\exists c) \\ & (\alpha(L_{cities}, \text{Cities, cityId}, c, 2006) = pg_1 \ \wedge\ c.name = \text{`Brussels'} \ \wedge \\ & \wedge\ \alpha(L_{land}, \text{Land, parcelId}, p, 2006) = g_{id} \ \wedge\ r^t_{L_{land}}(pg_1, e_p, 2006) \ \wedge \\ & r^t_{L_{land}}(g_{id}, e_g, 2006) \ \wedge\ distance(e_p, e_g) < 100)\} \end{aligned}$$

Here, we added a layer L_{cities}, linked to a dimension *Cities* by the α function. The variable c ranges over the dimension level cityId, and *c.name* represents the name of the city corresponding to the identifier that instantiates c when evaluating the expression (this is actually an attribute of the dimension level cityId). The problem of how the distance is computed is beyond the scope of this work, although it is easy to see that it is computable. The expression $\alpha(L_{land}, \text{Land, parcelId}, p, 2006) = g_{id}$ returns *true* if the function α maps the dimension member p in level parcelId, to the polygon g_{id} in layer L_{land} in 2006. The semantics is similar to the one explained in Example 2: the only free variable in C is g_{id}; thus, for a given instantiation of the quantified variables, C returns the values of the g_{id}'s in the database that make the formula *true*.

The next query includes topological relations, and temporal relationships. In the terms of Piet-QL, it is an OLAP query filtered with a GIS condition. Also, in this

query we assume a granularity of *month*, in order to show OLAP capabilities. We use the rollup function for this.

Q_2: Number of airports currently located in parcels that were adjacent to parcel $p3$ in November, 2004

This is expressed by:

$$Q_2 \equiv \sum_{g_{id} \in C} 1,$$

where C is defined by the expression:

$$C = \{g_{id} \mid (\exists t_1)(\exists p)(\exists p_1)(\exists e_a)(\exists e_p)(\exists e_{p_1})$$
$$(\alpha(L_{land}, \text{Land}, \text{parcelId}, \text{'p3'}, t_1) = p \wedge r^t_{L_{land}}(p, e_p, Now) \wedge r^t_{L_a}(g_{id}, e_a, Now) \wedge$$
$$r^t_{L_{land}}(p_1, e_{p_1}, t_1) \wedge f^{day \to month}_{Time}(t_1) = \text{'November'} \wedge f^{day \to year}_{Time}(t_1) = \text{'2004'} \wedge$$
$$adjacent(e_{p_1}, e_p) \wedge within(e_p, e_a))\}$$

The predicate $adjacent(e_{p_1}, e_p)$ returns *true* if e_{p_1} and e_p are instantiated with extensions that correspond to parcels adjacent to each other in November, 2004. Analogously, $within(e_p, e_a)$ returns *true* if an instantiation of e_p and e_a is such that currently the latter is inside the former. We also assume that 'p3' is the identifier of the parcel in the data warehouse (dimension level parcelId), so we used the mapping function to find the corresponding identifier of the spatial object in layer L_{land}, represented by the variable p. Note the use of the relations $r^t_{L_i}$. They allow to tell the state of each object at a given time; thus, the topological functions are exactly the same as in the non-temporal case. The terms $f^{day \to month}_{Time}$ and $f^{day \to year}_{Time}$ represent the rollup functions over the Time dimension in the warehouse. As a final observation, note that in case 'P3' would have been the identifier of the corresponding spatial object, we would not have needed the α function.

The next query includes a rollup function over an external warehouse, filtered by both, a GIS condition and aggregated data in an external fact table (in Piet-QL terminology).

Q_3: How many parcels currently located in the province of Antwerp, with more than 600mm of annual rain in 2006, are crossed by the Schelde river?

(to simplify presentation, we go back to the use of granularity *year*).

$$Q_3 \equiv \sum_{g_{id} \in C} 1.$$

$$C = \{g_{id} \mid (\exists p)(\exists p_1)(\exists r_{id})(\exists e_{r_{id}})(\exists e_{g_{id}})(\exists rain)$$
$$(\alpha(L_{land}, \text{Land}, \text{parcelId}, p, Now) = g_{id} \wedge \alpha(L_r, \text{Rivers}, \text{riverId}, p_1, Now) = r_{id} \wedge$$
$$p_1.name = \text{'Schelde'} \wedge r^t_{L_{land}}(g_{id}, e_{g_{id}}, 2006) \wedge r^t_{L_r}(r_{id}, e_{r_{id}}, 2006) \wedge$$
$$f^{parcelId \to province}_{Land}[Now](p) = \text{'Antwerp'} \wedge intersects(e_{g_{id}}, e_{r_{id}}) \wedge$$
$$WeatherFacts(p, 2006, rain) \wedge rain \geq 600)\}$$

Here, the expression $f_{Land}^{parcelId \rightarrow province}[Now](p) =$ 'Antwerp' represents the rollup function in dimension *Land*, such that it returns *true* if parcel p currently rolls up to the province of Antwerp. There is a fact table in the warehouse, denoted *WeatherFacts* in the warehouse, with measure *rain* (the annual average precipitation).

The next one is a non-aggregate query, using only the relations $r_{L_i}^t$.

Q_4: Parcels having an area larger than 100 Ha in 1996, and currently larger than they were at that time.

$$Q_4 = \{p \mid (\exists e_p)(\exists e_{p_1})(\exists a) \\ (r_{L_{land}}^t(p, e_{p_1}, 1996) \wedge r_{L_{land}}^t(p, e_p, Now) \wedge \\ area(e_{p_1}) = a \wedge a > 100 \wedge area(e_p) > a)\}.$$

This query does not mention warehouse data. Here, $r_{L_{land}}^t$ holds the parcels existing in 1996 and today, and their areas. This relation refers to two different time instants, indicated as constants (note that variable p is used in both cases). The function *area* is similar to the function *extent* in postGIS.

The second motivating example in Section 1 allows to show some more involved temporal queries. We want to assess the effect of the polluting clouds over people. Thus, we are interested to know how many people were exposed to this cloud for some time. For this, we combine trajectory information and moving regions. Trajectories are given as samples of the form (O_{id}, x, y, t), stating that an object O_{id} is at the coordinate x, y at an instant t, stored, as we explained above, in a table denoted MOFT (standing for Moving Object Fact Table, although this is a special kind of -measureless- fact table). We assume that the trajectory of the cloud is recorded as a 'temporal' layer $r_{L_c}^t$. For example, satellite images are recorded at regular intervals and stored in a thematic layer.

Q_5: Objects whose trajectories had passed below a polluting cloud

$$\{O_{id} \mid (\exists c)(\exists e_c)(\exists t)(\exists x)(\exists y) \\ (r_{L_c}^t(c, e_c, t) \wedge \\ MOFT(O_{id}, x, y, t) \wedge within(\langle x, y\rangle, e_c))\}$$

Q_6: Objects that remained below a polluting cloud for more than fifteen consecutive minutes.

$$\{O_{id} \mid (\exists c)(\exists e_c)(\exists c_1)(\exists e_{c_1})(\exists c_2)(\exists e_{c_2}) \\ (\exists t_1)(\exists t_2)(\exists x)(\exists y)(\exists x_1)(\exists y_1)(\exists x_2)(\exists y_2) \\ (r_{L_c}^t(c, e_{c_1}, t_1) \wedge r_{L_c}^t(c_2, e_{c_2}, t_2) \wedge \\ MOFT(O_{id}, x_1, y_1, t_1) \wedge MOFT(O_{id}, x_2, y_2, t_2) \wedge \\ \wedge within(\langle x_1, y_1\rangle, e_{c_1}) \wedge within(\langle x_2, y_2\rangle, e_{c_2}) \\ \neg(\exists t)\neg(\exists c)\neg(\exists e_c)(r_{L_c}^t(c, e_c, t) \wedge \neg(within(\langle x, y\rangle, e_c))) \wedge \\ t_2 \geq t1 + 15 \wedge t_1 \leq t \leq t_2)\}$$

Note that we need three variables for the object c, because the object can change between t_1 and t_2. The only free variable is O_{id}. Since we use the notion of *active domain* [1], x_i, y_i and t_i are instantiated only with values in the MOFT.

Complexity Although a thorough study of complexity of $\mathcal{L}_t$ is a topic of further research, we now briefly comment on this issue. A particularity of our language (which we characterized according to [66] as falling in-between the ST-OLAP and ST-TOLAP classes of queries), is that it involves aggregation, spatial objects, and temporal variables. Thus, we start analyzing known results in the field of query languages. First, $\mathcal{L}_t$ could be seen as a FO language over $\mathbb{R}$ with the addition of aggregation, spatial data and topological relations, and it has been proved that FO queries have data complexity QLOGSPACE [1]. With respect to the impact of aggregation, it is well-known that non-recursive stratified datalog (SNR-DATALOG) expresses exactly FO queries, and in [12] it is shown that SNR-DATALOG extended with aggregate functions (SNR-DATALOG$^{\mathrm{AGG}}$) is equivalent to the relational algebra with aggregate functions defined by Klug [31] (in turn, equivalent to relational calculus with aggregate functions), denoted FO$^{\mathrm{AGG}}$ (the calculus over which the classification in [66] is built). The authors show that FO$^{\mathrm{AGG_L}} \subseteq$ QLOGSPACE, where $\mathrm{AGG_L}$ is the set of aggregate functions such that the decision problem associated with the evaluation of the aggregate operations is in LOGSPACE (the set AGG, composed of the five classic aggregate functions MAX, MIN, SUM, COUNT, AVG, is strictly contained in $\mathrm{AGG_L}$). In light of these discussion, the complexity of $\mathcal{L}_t$ can be assessed as the complexity of FO$^{\mathrm{AGG_L}}$ affected by the impact of computing the binary topological relations r_{top}.

7 Implementation, updates and TPiet-QL

In this section we sketch how the ideas in the previous sections are implemented in Piet. We first give an overview of the data structure of Piet, and then we discuss how we add temporal support.

7.1 Piet data structure

The data structure for the current Piet implementation is organized in:

- Application information. This is the data warehouse structure, which can be navigated with OLAP tools.
- GIS information. Corresponds to the data structure for the map layers, including the spatial (geometric) objects in each layer, and associated attributes.
- 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).
- There are also data structures to store precomputed information containing the overlay of different layers. We do not discuss this issue in this paper. The interested reader is referred to [16].

In Fig. 7, the "Application Information" part contains the warehouse dimensions and fact tables (we show portions of schemas of the dimensions *Land* and *Airports*,

Fig. 7 Piet data structure

and the fact table *WeatherFacts*). The "GIS Information" part contains a table for each layer in the system (we show *Land*, *Rivers* and *Airports*). Each table schema has, at least, a unique layer identifier, and the extension of the geometric objects. There is additional information (not shown in the picture, since we do not use it for the temporal extension) used to compute the overlay when needed. The tables also contain non-spatial attributes (e.g., area). Finally, there is the GIS-OLAP mapping part. There, elements in the OLAP dimensions are matched with elements in the GIS. This matching is performed in a semi-automatic fashion. First, a program based on weighted bipartite matching ir run. A curator should later verify the accuracy of this matching, and make the necessary changes.

7.2 Implementing the temporal extension: updates

The system must be able to manage changes to the geometric objects. We are aimed at supporting the following changes, usually found in the literature [47] (a full study of the implementation of these operators is outside the scope of this paper):

- Creation;
- Update (e.g., a change in the object's extension, or in the object's location in space);
- Deletion (in the temporal sense, i.e., without losing the object's history);
- Reincarnation (an object is deleted, and then re-created)
- Split (an object generates two or more new ones, and then it is deleted).
- Merge (two or more objects are merged into a new one, and deleted)

The different states 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 how the operators above are implemented. 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 t_1 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.

Note that we have not said anything yet about 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 here: (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 last state of the dimension data is available. In other words, the data warehouse is *non-temporal*. (b) A *temporal* warehouse is available, i.e., dimensions are updated and their history is preserved. We adopt here the concept of *slowly changing dimensions* [29], 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 [14, 36, 37]. In the former case, the expressive power of the query language may be reduced, as we explain below (Section 7.3). In both cases, some the update operations described above should also include the update of the mapping tables between the GIS and the OLAP parts, and consistency checking against the warehouse. This occurs when the operations require creating new identifiers. For example, in the *creation, split,* and *merge* operations, new objects are introduced in the GIS; thus, the objects should be inserted in the warehouse dimensions and new mappings must be defined. However, note that an *update* has no impact on the data warehouse or in the mapping, since the object identifier does not change in this case. 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).

7.3 Temporal extension to Piet-QL

We now sketch an extension to the Piet-QL query language, based on the formal language of Section 6. We denote this extension TPiet-QL.

Let us start with a brief overview of Piet-QL (see [18] for a detailed study). We present the flavor of the language through an example. We have already said that Piet-QL supports four basic kinds of queries: (a) standard GIS queries; (b) standard OLAP queries; (c) GIS queries, filtered with an aggregation (i.e., filtered using a data cube); (d) OLAP queries filtered with a GIS condition. To begin with, let us

show an example of a Piet-QL query of type (d). There is a data cube (denoted Sales), containing the aggregation (at different granularity levels) of the sales of the production of the parcels in different regions in Belgium. There is also a layer containing the cities in Belgium. Consider now the following query: "Production cost and parcel sales, for parcels crossed by rivers". This query reads, in Piet-QL:

```
SELECT CUBE [Measures].[Production Cost],
[Measures].[Parcel Sales] ON COLUMNS,
Product.[All_Products] ON ROWS
FROM [Sales]
WHERE [Land].[All Land] IN(
  SELECT GIS land
  FROM land, river
  WHERE intersects(river,land);
```

This query returns, for each parcel and product, the corresponding production costs and sales of the parcels that verify the geometric condition. The result can be navigated in the usual OLAP fashion. The FROM clause of the outer query indicates the data cube (*Sales*) from which aggregated non-spatial information is extracted. In the inner query, *land* and *river* are GIS layers. Now, let us suppose that we have stored the history of the parcels and the rivers. We extend Piet-QL to be able to query such history. This way, the query: "Production cost and parcel sales, for the parcels crossed by rivers in 2005", would read in TPiet-QL:

```
SELECT CUBE [Measures].[Production Cost],
[Measures].[Parcel Sales] ON COLUMNS,
Product.[All_Products] ON ROWS
FROM [Sales]
WHERE [Land].[All Land] IN(
  SELECT GIS land
  FROM land, river
  WHERE intersects(river,land) AND
  land.FROM < 2005 AND land.TO > 2005 AND
  river.FROM < 2005 AND river.TO > 2005;
```

Note the similarity between this query and the expressions in the formal query language of Section 6.

Let us now consider the following query, of type (c): "Parcels crossed by the Dÿle river, with production sales greater than 5,000 units" (i.e., a GIS query, filtered by aggregated values). In Piet-QL the query reads:

```
SELECT GIS l.id
FROM land l, rivers lr
WHERE intersects(l, lr) AND lr.name = "Dÿle"
AND l.interval overlaps lr.interval
AND l IN(
  SELECT CUBE
  filter([Land].[Land parcelId].Members,
```

```
[Measures].[Parcel Sales] > 5000)
FROM [Sales]);
```

In this query, *l* and *lr* range over the elements in the layers. For each layer identifier in the GIS, the subquery is run over the data cube. We have included the *overlaps* interval operator, but any interval operation defined in the seminal work of Allen [4] can be used. Since the predicate *intersects* is time-independent, the interval operator defines the semantics of the intersection (e.g., in the query above, intersects will return true if the river crossed the parcel at least during some time interval). Also note that, for the inner query, the semantics implies that all the history is accounted for, meaning for example, that if there are several tuples for the same parcel identifier (due to successive updates) all of them will be checked against the data cube.

Note that if dimension updates were supported in the warehouse, more powerful queries could be supported, meaning that temporal queries could be posed in the inner query above, along the lines of [36, 37]. Otherwise, queries cannot include temporal conditions over the warehouse dimensions. However, temporal conditions in queries just involving the dimension identifiers can be expressed in TPiet-QL, even if the warehouse does not support temporal semantics. More precisely, let us assume a parcel p_1 was created in 1995, and split in 2004 into p_{11} and p_{12}. Production sales were recorded for all these years in a fact table, at the level of parcel identifier, i.e., tuples in this fact table are of the form $\langle p_1, 1995, 1000\rangle$,... $\langle p_{11}, 2007, 800\rangle$,..$\langle p_{12}, 2007, 900\rangle$.

Querying moving object data, like in Q5 and Q6 in Section 6, requires that the language supports external tables (currently this is not handled by Piet-QL). For example, "Objects (and the instants) whose trajectories had passed below a polluting cloud" (query Q5 in Section 6), would read in TPiet-QL:

```
SELECT DISTINCT m.Oid,m.t
FROM MOFT m, GIS cloud c,
WHERE contains(c,m.p) AND
m.t ≥ c.FROM AND m.t ≤ c.TO;
```

Recall that the structure of the MOFT is $(O_{id}, point, t)$, where point indicates the x, y coordinates of the object's position at time t. Variable m ranges over the MOFT (Moving Object Fact Table) and c ranges over the objects is the layer *cloud*.

7.4 Spatio-temporal integrity constraints

There exists extensive literature on the topic of integrity constraints in spatial databases. Cockcroft [11] proposed a taxonomy for these kinds of constraints that also includes the traditional (non-spatial) referential integrity constraints present in relational database systems. The author defines a hierarchy of spatial integrity constraints which includes the traditional ones, plus the new classes: topological, semantic and user-defined constraints. The topological class includes the constraints studied by Hadzilacos and Tryfona [23]. Semantic constraints in this taxonomy are concerned with the meaning of geographical features. An example is a a constraint stating that a user cannot enter a road running through a lake. Finally, user-defined constraints resemble more classic database rules. For example, a rule

can state that gas stations cannot be located closer than a given distance from a school. The author also suggests technologies that could be used to implement the spatial constraints. Later, Tryfona and Hadzilacos [61] introduce a spatio-temporal data model (denoted STRM) that includes a language to define spatio-temporal integrity constraints, based in a predicate calculus that includes atomic formulas, negation conjunction, disjunction, and (universal and existential) quantification. This allows expressing complex spatiotemporal integrity constraints, using a DEFINE CONSTRAINT statement. The statement includes the layers and objects involved, and the specification of the constraint as an FO-formula. Borges et al. [9] propose OMT-G, an extension of the OMT model for geographic applications for the specification of integrity constraints in spatial databases. The model provides a collection of primitives for representing spatial data, supporting topological, semantic and user-defined integrity rules to be specified in the database schema, based on the taxonomy defined by Cockcroft. More recently, Rodriguez et al. [54] define a *repair* semantics for spatial databases with respect to spatial semantic integrity constraints which impose semantics on topologies, in particular for binary spatial predicates that represent topological relations between regions. In short, a repair semantics defines admissible and consistent alternative instances to for an inconsistent spatial database.

The formal language we propose in this paper and its implementation TPiet-QL, can be used to specify constraints not only over the GIS layers, but also over the data warehouse. STRM defines constraints as FO-formulas, like $\mathcal{L}_t$ does. In addition, the Piet data model (and its associated language $\mathcal{L}_t$) supports GIS-OLAP integration, a feature not present in other proposals. Thus, it would be straightforward to extend TPiet-QL (which we already commented is strongly based on $\mathcal{L}_t$) to be a constraint definition language (note that this is a concept already used in relational databases where SQL is used a a basis for defining triggers).

8 Conclusion and future work

Based on an existing formal data model (Piet) we have discussed two ways of introducing time into SOLAP. We first present a modified Piet conceptual model, keeping its features while making it simpler and closer to the implementation layer. Then, we studied two ways of adding temporal semantics to Piet, based on two temporal database paradigms: the *snapshot* and the *timestamp* data models. We developed a formal model for each of these alternatives and discussed their features, showing that the timestamp model better fulfills the SOLAP requirements. We then defined a temporal First-Order query language, providing comprehensive examples that illustrate its use. We finally moved on to implementation issues. First, we discussed how the data structure of Piet is modified to support temporality. Second, how this structure supports typical updates on the GIS side and the implications of the existence or not of a warehouse with temporal semantics. Finally, we sketched an extension of Piet-QL (the query language of Piet), with temporal semantics.

Future work regarding TPiet-QL includes an ongoing real world case study over river pollution in the cost of Uruguay, and the study of query optimization techniques. In particular, we are planning to study mechanisms that can take advantage of the fact that historical information can be aggregated and stored as materialized

views that will not have to be further updated as new information arrives. Techniques of this kind have been applied in temporal data warehousing [64], and we think that some of these ideas could be adapted to be applied in spatio-temporal query processing. Finally, extending the language to allow the definition of constraints as explained in Section 7.4 is another promising line of work.

Open research issues We commented in Section 2 that the contribution of the GIS and OLAP communities to the study of models involving together *OLAP and continuous fields* has been rather limited so far. Given this, and given that the study of continuous fields is increasingly gaining attention from the GIS practitioners, we believe that integrating continuous fields and OLAP applications is a relevant research topic. In this sense, one of our future research directions is oriented to extend TPiet-QL to support not only spatial objects as we discussed in the present paper, but continuous fields as well. This requires extending the data model and the formal query language supporting TPiet-QL, and the implementation of the latter to support fields. One possible approach could be to add a new *field* data type (and its associated *lifted* moving type *à la* Güting et al. [21]). This is the approach proposed by Vaisman and Zimányi [66], who proposed a conceptual model for multidimensional analysis of continuous fields. This work extends the classification of spatial OLAP queries proposed in [65] with two new query classes: SOLAP-CF and STOLAP-CF. The former is the class containing the queries that can be expressed using the Klug's relational calculus [31] augmented with spatial and field types. The latter is the class containing the queries that can be expressed using the Klug's relational calculus augmented with spatial types, field types, and moving spatial types.

There is also room for research on the theoretical side, as commented in Section 6. A thorough study of the complexity and expressiveness of the formal language presented in this paper has not yet been performed. For this, theoretical results in spatial databases (see for example [42] and [43]) can be applied and extended.

References

1. Abiteboul S, Hull R, Vianu V (1995) Foundations of databases. Addison-Wesley, Reading, MA
2. Ahmed TO, Miquel M (2005) Multidimensional structures dedicated to continuous spatiotemporal phenomena. In: 22nd British national conference on databases (BNCOD). Sunderland, UK, pp 29–40
3. Ahmed TO (2008) Continuous spatial data warehousing. In: International Arab conference on information technology
4. Allen J (1983) Maintaining knowledge about temporal intervals. Commun ACM 26(11):832–843
5. Armstrong MP (1988) Temporality in spatial databases. In: GIS/LIS. San Antonino, Texas, USA, pp 880–889
6. Bédard Y, Merret T, Han J (2001) Fundamentals of spatial data warehousing for geographic knowledge discovery, chap 3. Taylor & Francis, pp 53–73
7. Bédard Y, Rivest S, Proulx MJ (2007) Spatial Online Analytical Processing (SOLAP): concepts, architectures, and solutions from a geomatics engineering perspective, chap 13. IGI Global, pp 298–319
8. Bettini C, Dyreson C, Evans W, Snodgrass R, Sean Wang X (1998) A glossary of time granularity concepts. Temporal databases: research and practice. Lect Notes Comput Sci 1399:406–411
9. Borges C, Laender AHF, Davis CA (1999) Piet-QL: spatial data integrity constraints in object oriented geographic data modeling. In: 7th ACM SIGSPATIAL international symposium on advances in geographic information systems (ACM-GIS). Kansas City, USA, pp 1–6

10. Cabibbo L, Torlone R (1997) Querying multidimensional databases. In: Database programming languages. Lecture notes in computer science, vol 1369. Springer, pp 319–335
11. Cockcroft S (1997) A taxonomy of spatial data integrity. Geoinformatica 1(4):327–343
12. Consens M, Mendelzon A (1990) Low complexity aggregation in graphlog and datalog. In: Third international conference on database theory (ICDT). Paris, France, pp 379–394
13. Dyreson C, Evans W, Lin H, Snodgrass R (2000) Efficiently supporting temporal granularities. IEEE transactions on data and knowledge engineering (TKDE), vol 12(4), pp 568–587
14. Eder J, Koncilia C, Morzy T (2002) The comet metamodel for temporal data warehouses. In: CAiSE. Toronto, Canada, pp 83–99
15. Egenhofer M, J. Herring J (1991) Categorizing binary topological relationships between regions, lines, and points in geographic databases. Technical report, Department of Surveying Engineering, University of Maine
16. Escribano A, Gomez L, Kuijpers B, Vaisman AA (2007) Piet: a gis-olap implementation. In: ACM 10th international workshop on data warehousing and OLAP (DOLAP). ACM, pp 73–80
17. Gómez L, Kuijpers B, Vaisman AA (2008) Aggregation languages for moving object and places of interest. In: ACM symposium on applied computing SAC. Fortaleza, Ceara, Brazil, pp 857–862
18. Gómez L, Vaisman A, Zich S (2008) Piet-QL: a query language for GIS-OLAP integration. In: 16th ACM SIGSPATIAL international symposium on advances in geographic information systems (ACM-GIS), 27. Irvine, CA, USA
19. Gómez L, Haesevoets S, Kuijpers B, Vaisman AA (2009) Spatial aggregation: data model and implementation. Inf Syst 34(6):551–576
20. Gray J, Bosworth A, Layman A, Pirahesh H (1997) Data cube: a relational operator generalizing group-by, cross-tab and sub-totals. Data Mining and Knowledge Discovery (1):29–53
21. Güting RH, Böhlen M, Jensen C, Lorentzos N, Schneider M, Vazirgiannis M (2000) A foundation for representing and quering moving objects. ACM Trans Database Syst 25(1):1–42
22. Güting RH, de Almeida VT, Ansorge D, Behr T, Ding Z, Höse T, Hoffmann F, Spiekermann M, Telle U (2005) SECONDO: an extensible dbms platform for research prototyping and teaching. In: 21st international conference on data engineering (ICDE). Tokyo, Japan, pp 1115–1116
23. Hadzilacos T, Tryfona N (1996) Logical data modelling for geographical applications. Int J Geogr Inf Sci 10(2):179–203
24. Han J, Stefanovic N, Koperski K (1998) Selective materialization: an efficient method for spatial data cube construction. In: Research and development in knowledge discovery and data mining (PAKDD). Lecture notes in computer science, vol 1394. Springer, pp 144–158
25. Hurtado C, Mendelzon A, Vaisman A (1999) Maintaining data cubes under dimension updates. In: 15th international conference on data engineering (IEEE/ICDE). Sydney, Australia, pp 346–355
26. Hurtado CA, Mendelzon AO (2001) Reasoning about summarizability in heterogeneous multidimensional schemas. In: International conference of database theory (ICDT), pp 375–389
27. Hurtado CA, Mendelzon AO (2002) OLAP dimension constraints. In: 21st ACM SIGACT-SIGMOD-SIGART symposium on principles of database system (PODS). Madison, Wisconsin, USA, pp 169–179
28. Kemp KK (1997) Fields as a framework for integrating gis and environmental process models. Trans GIS 1(3):219–246
29. Kimball R (1996) The data warehouse toolkit. Wiley, New York
30. Kimball R, Ross M (2002) The data warehouse toolkit: the complete guide to dimensional modeling, 2nd edn. Wiley, New York
31. Klug A (1982) Equivalence of relational algebra and relational calculus query languages having aggregate functions. J ACM 29(3):699–717
32. Kuijpers B, Vaisman A (2007) A data model for moving objects supporting aggregation. In: Proceedings of the first international workshop on spatio-temporal data mining (STDM'07). Istambul, Turkey
33. Langran G, Chrisman NR (1988) A framework for temporal geographic information systems. Cartographica 25(3):1–14
34. López IFV, Snodgrass R, Moon B (2005) Spatiotemporal aggregate computation: a survey. IEEE Trans Knowl Data Eng 17(2):271–286
35. Malinowski E, Zimányi E (2004) Representing spatiality in a conceptual multidimensional model. In: 12th ACM international workshop on geographic information systems(GIS). Washington, DC, USA, pp 12–22

36. Mendelzon AO, Vaisman AA (2000) Temporal queries in OLAP. In: 26th international conference on very large data base (VLDB). Cairo, Egypt, pp 242–253
37. Mendelzon AO, Vaisman AA (2003) Time in multidimensional databases. In: Multidimensional databases. IDEA Group, pp 166–199
38. Orlando S, Orsini R, Raffaetà A, Roncato A, Silvestri C (2007) Spatio-temporal aggregations in trajectory data warehouses. In: Data warehousing and knowledge discovery (DaWak). Lecture notes in computer science, vol 4654. Springer, Regensburg, Germany, pp 66–77
39. Paolino L, Tortora G, Sebillo M, Vitiello G, Laurini R (2003) Phenomena: a visual query language for continuous fields. In: 11th ACM international workshop on geographic information systems (GIS). New Orleans, LA, USA, pp 147–153
40. Papadias D, Kalnis P, Zhang J, Tao Y (2001) Efficient OLAP operations in spatial data warehouses. In: Advances in spatial and temporal databases (SSTD). Lecture notes in computer science, vol 2121. Springer, pp 443–459
41. Papadias D, Tao Y, Kalnis P, Zhang J (2002) Indexing spatio-temporal data warehouses. In: 18th international conference on data engineering (ICDE). San Jose, CA, USA, pp 166–175
42. Papadimitriou CH, Suciu D, Vianu V (1996) Topological queries in spatial databases. In: Proceedings 15th ACM SIGACT-SIGMOD-SIGART symposium on principles of database system (PODS). Montreal, Canada, pp 81–92
43. Paredaens J, den Bussche JV, Gucht DV (1994) Towards a theory of spatial database queries. In: 13th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, (PODS). Minneapolis, USA, pp 279–288
44. Paredaens J, Kuper G, Libkin L (eds) (2000) Constraint databases. Springer, Berlin Heidelberg New York
45. Pedersen TB, Tryfona N (2001) Pre-aggregation in spatial data warehouses. In: Advances in spatial and temporal databases (SSTD), pp 460–480
46. Pelekis N (2002) Stau: a spatio-temporal extension to ORACLE DBMS. Ph.D thesis, UMIST Department of Computation
47. Pelekis N, Theodoulidis B, Kopanakis Y, Theodoridis Y (2004) Literature review of spatio-temporal database models. The Knowledge Engineering Review Journal 19(3):235–274
48. Pelekis N, Theodoridis Y, Vosinakis S, Panayiotopoulos T (2006) Hermes—a framework for location-based data management. In: 10th international conference on extending database technology. Munich, Germany, pp 1130–1134
49. Peuquet D, Duan N (1995) An event-based spatiotemporal data model (estdm) for temporal analysis of geographical data. Int J Geogr Inf Syst 9(1):7–24
50. Pourabbas E (2003) Cooperation with geographic databases. In: Multidimensional databases: problems and solutions. Idea group, pp 393–432
51. Rao F, Zhang L, Yu X, Li Y, Chen Y (2003) Spatial hierarchy and OLAP-favored search in spatial data warehouse. In: ACM 6th international workshop on data warehousing and OLAP (DOLAP). New Orleans, LA, USA, pp 48–55
52. Rigaux P, Scholl M, Voisard A (2001) Spatial databases: with application to GIS. Data management systems. Morgan Kaufmann, San Mateo, CA
53. Rivest S, Bédard Y, Marchand P (2001) Towards better support for spatial decision making: defining the characteristics of spatial on-line analytical processing (SOLAP). Geomatica 55(4):539–555
54. Rodriguez A, Bertossi L, Caniupan M (2008) An inconsistency tolerant approach to querying spatial databases. In: 16th ACM SIGSPATIAL international symposium on advances in geographic information systems (ACM-GIS), 36. Irvine, CA, USA
55. Shanmugasundaram J, Fayyad UM, Bradley PS (1999) Compressed data cubes for olap aggregate query approximation on continuous dimensions. In: 5th ACM SIGKDD international conference on knowledge discovery and data mining (KDD). San Diego, CA, USA, pp 223–232
56. Shekhar S, Lu CT, Tan X, Chawla S, Vatsavai RR (2001) Map Cube: a visualization tool for spatial data warehouses, chap 4. Taylor and Francis, pp 73–108
57. Snodgrass R (ed) (1995) The TSQL2 temporal query language. Kluwer, Boston, MA
58. Stefanovic N, Han J, Koperski K (2000) Object-based selective materialization for efficient implementation of spatial data cubes. IEEE Trans Knowl Data Eng 12(6):938–958
59. Tansel A, Clifford J, Gadia S (eds) (1993) Temporal databases: theory, design and implementation. Benjamin Cummings, Redwood City, CA
60. Toman D (1996) Point vs. interval-based query languages for temporal databases. In: 15th ACM SIGACT-SIGMOD-SIGART symposium on principles of database system (PODS). Montreal, Canada, pp 58–67

61. Tryfona N, Hadzilacos Th (1998) Logical data modelling of spatio temporal applications: definitions and a model. In: International database engineering and applications symposium(IDEAS). Cardiff, Wales, UK, pp 14–23
62. Tryfona N, Jensen C (1999) Conceptual data modeling for spatiotemporal applications. Geoinformatica 3(3):245–268
63. Tryfona N, Price R, Jensen C (2003) Conceptual models for spatiotemporal applications. Spatiotemporal databases: the CHOROCHRONOS approach, pp 79–116
64. Vaisman AA, Izquierdo A, Ktenas M (2008) A web-based architecture for temporal olap. Int J Web Eng Technol 4(4):465–494
65. Vaisman A, Zimányi E (2009) What is spatiotemporal data warehousing? Data warehousing and knowledge discovery (DaWak). In: Proceedings of DaWaK, pp 9–23
66. Vaisman A, Zimányi E (2009) A multidimensional model representing continuous fields in spatial data warehouses. In: Proceedings of ACM-SIGSPATIAL, pp 168–177
67. Wachowicz M, Healey R (1994) Towards temporality in GIS. In: Innovation in GIS, vol I. Taylor & Francis, pp 105–115
68. Worboys MF (1992) A model for spatio-temporal information. In: Proceedings of the 5th international symposium on spatial data handling. Charleston, South Carolina, pp 602–611
69. Worboys MF (1995) GIS: a computing perspective. Taylor & Francis, New York
70. Zeiler M (1999) Modeling our world: the ESRI guide to geodatabase design. ESRI Press
71. Zhang L, Li Y, Rao F, Yu X, Chen Y, Liu D (2003) An approach to enabling spatial OLAP by aggregating on spatial hierarchy. In: Data warehousing and knowledge discovery (DaWak). Lecture notes in computer science, vol 2737. Springer, Prague, Czech Republic, pp 35–44