VII Simpósio Brasileiro de Geoinformática, Campos do Jordão, Brasil, 20-23 novembro 2005, INPE, p. 155-173.

The Architecture of a Flexible Querier for Spatio-Temporal Databases

Karine Reis Ferreira, Lúbia Vinhas, Gilberto Ribeiro de Queiroz, Ricardo Cartaxo Modesto de Souza, Gilberto Câmara

Divisão de Processamento de Imagens – Instituto Nacional de Pesquisas Espaciais (INPE)
Av. dos Astronautas, 1758 – 12220-010 – São José dos Campos – SP – Brazil
{karine, lubia, gribeiro,cartaxo,gilberto}@dpi.inpe.br

Abstract. *In this paper, we propose a solution to the problem of designing a flexible query processor for spatio-temporal databases. Our solution is to design a bottom-up interface, whose parameters handle different spatio-temporal applications. We propose a relational schema that can cope with various spatio-temporal data types. Based on this model, we have design a flexible query processor for spatio-temporal databases. We have implemented this query processor in TerraLib open source GIS environment and we present a case study.*

1. Introduction

Emerging database management systems that can handle spatial data types have changed both GISystems and GIScience. Systemwise, this technology enables a transition from the current GIS technology to a new generation of *spatial information appliances*, tailored to specific user needs [Egenhofer 1999]. For the GIScience community, it enables many theoretical proposals to face the crucial test of practice. One of the important challenges for the GIScience community is finding ways to use spatially enabled DBMS to build innovative applications which deal with spatio-temporal data [Erwig, Güting et al. 1999] [Hornsby and Egenhofer 2000]. Modeling spatio-temporal applications is a complex task that involves representing objects with spatial extensions and attributes values that change over time [Frank 2003]. To deal with spatio-temporal data, one alternative is building a specialized DBMS created for efficient support of spatio-temporal data types, as in the projects CONCERT [Relly, Schek et al. 1997] and SECONDO [Dieker and Güting 2000]. When is not possible to use a specialized DBMS, one has to build a layered architecture on top of an existing object-relational DBMS. This is the focus of this paper, where we consider how to support applications of spatio-temporal data, using object-relational database management systems (OR-DBMS). In this case, one basic question arises: *how to design a flexible query processor for spatio-temporal data using object-relational DBMS?*

A flexible query processor needs to be able to cope with different applications of spatio-temporal data and their needs for queries and responses. To solve this problem, a popular approach in the literature is to provide specialized algebras for different applications. For example, Güting, Bohlen et al. [2003] present a model for moving objects that includes moving points and moving regions. Hornsby and Egenhofer [2000] and Medak [2001] propose models for the life and evolution of socio-economic objects.

These specialised models can lead to databases where each type of application is handled by a different query processor. Obviously, this is not desirable for developers of applications using spatio-temporal databases. Ideally, the architecture of the query processor would have a unified and flexible way of dealing with the different applications of spatio-temporal data.

In this paper, we propose a solution to the problem of designing a flexible query processor for spatio-temporal databases using object-relational DBMS. Our proposed solution is to unify the internal architecture of the database for all different spatio-temporal applications. In this paper, we discuss the design and architecture of the query processor. In section 2, we review the issue of spatio-temporal query processing. In section 3, we present the database model and the architecture of the query processor. In section 4, we show a case study using the query processor.

2. Spatio-temporal data handling: top-down x bottom-up approaches

In this section, we discuss four different levels in a database design that address different aspects for handling of spatio-temporal data: (a) A set of data types and an associated algebra; (b) A conceptual data model for spatio-temporal data; (c) A spatio-temporal query language; (d) An application programming interface with suitable parameters. We consider the first three as "top-down" approaches and the fourth as a "bottom-up" choice.

The first alternative is to define a set of spatio-temporal data types and operators. The DBMS is extended to support these data types and operators and will provide an associated query language. This is the approach taken by Güting [2005] that defines an algebra for moving objects. His spatio-temporal data types for moving objects are embedded in a query language to answer queries as: *"Given the trajectories of two airplanes, when they will pass over the same location?*". Similarly, Medak [2001] proposes an algebra for modeling change in socio-economical units. Medak's algebra provides answers to queries such as: *"When was this parcel divided?"* The main challenge of this approach is finding a suitably small set of data types and operators for handling all types of spatio-temporal data. Currently, we only find spatio-temporal algebras for specialized applications (e.g., moving objects).

The second choice is to design a conceptual model for spatio-temporal data. In this case, the designer starts from an external view of the problem and provides a set of classes (or an equivalent E-R model). These classes encapsulate abstractions such as geometry, attributes and their changes. Examples include STER [Tryfona and Jensen 1999] and MADS [Parent, Spaccapietra et al. 1999] (see also Pelekis et al [2004] for a review of similar models). The main drawback of these approaches is the large variety of different application semantics for combining space and time. These models work fine for some applications, but will not fit other cases well.

The third approach is the design of a general spatio-temporal query language, which needs a well-defined set of predicates for spatial, temporal, and spatio-temporal queries. For spatial data, topological and directional operators are already well-established in the literature [Egenhofer and Franzosa 1991] [Papadias and Egenhofer 1997] [Clementini and Di Felice 1996]. Dealing with temporal data is also a well-researched issue. The interval algebra for temporal operators is established [Allen

1983], as is the bitemporal model of Worboys [1994]. However, there are problems when trying to devise a unique canonical set of spatio-temporal predicates. As shown by Erwig and Schneider [2002], it is not practical to devise one such set because there are too many predicates that can be considered different. They propose two options. Either each application will develop a specialized subset of predicates, or the spatio-temporal database will provide *combinators* that allow the user to build up her or his own predicates.

Given the lack of generality of these approaches, we have taken a fourth route. Our bottom-up alternative is to design a query processor as a parametrizable function. Taking in consideration the suggestion by Erwig and Schneider [2002] to design a combinator, this query processor is flexible and can be used by different applications. The set of parameters of the query processor works as a combinator of spatio-temporal predicates. We discuss this query processor in the next sections of the paper. Since our proposal aims at a generic way of dealing of spatio-temporal queries, we do not discuss query optimization in this paper. We consider the main contribution of the work is to provide a programming interface that can be optimized later for handling specific applications.

3. The architecture of a spatio-temporal query processor

3.1. General view of a spatio-temporal database

In this section, we describe a generic model for a geographical database, which is the basis for designing the query processor. We assume that a geographical database stores *layers*. A *layer* aggregates spatial information that covers a geographical region and has a common set of attributes and shares the same spatial reference system. Layers supports both the object-based and field-based models of spatial information [Couclelis 1992]. The layer model is used by most spatial extensions of object-relational DBMS such as ORACLE SPATIAL and PostGIS. In this work we concentrate on the object-based layers.

Figure 1 shows a layer of districts of the Brazilian city of Recife. Each district has a set of descriptive attributes, such as its name, or the population of the district in the census of 2004. The spatial extension of each district is a polygon that represents its boundaries.

ID_	NAME	POP_2004
2611606050001	Recife	98361
2611606050010	Cabanga	102548
2611606050011	Sao Jose	1524
2611606050012	Derby	157621
2611606050013	Gracas	278951
2611606050014	Jaqueira	102200

Figure 1. A layer of districts of Recife, PE.

Our model considers that a layer contains a set of *spatio-temporal objects (ST-Objects)*. An *ST-Object* is an entity that preserves its identity over time [Hornsby and Egenhofer 2000]. *Static layers* aggregate *ST-Objects* with geometry and attribute values that do not change. *Temporal layers* aggregate *ST-Objects* that change their attribute values or their geometry. We refer to the different versions of the same *ST-Object* as *spatio-temporal instances (ST-Instances)*. Each *ST-Instance* has an associated interval that is the validity time of that instance and knows its current spatial extension and its current set of attribute values.

3.2. A generic database model for spatio-temporal data

3.2.1 Static Layers

Our database model considers in a set of relations that include attribute relations, geometry relations and metadata relations. In a generic way, we represent the geometry and attribute relations as:

```
geometries(geomId:int, objId:string, spatialData: spatial)
attributes(objId:string, [attl:attType,...,attn:attType]).
```

We consider that each entity has a unique and persistent identification (`objId`). Each geometry also has a unique identifier (`geomId`). We use `attType` for conventional types such as `int`, `double` or `string`. We use the `spatial` keyword for types that can store a spatial extent. We consider these relations as *data relations* since they effectively store the spatial data.

We also need to store metadata information on the database. These relations describe the geometry and attributes relations associated to each layer. Our metadata relations are:

```
layers          (layerId: int, layerName: string)
representations (layerId: int, geomRelation: string)
attributesRel   (layerId: int, attrRelation: string)
```

The `layers` relation provides a unique identifier (`layerId`) for each layer. It can also contains other attributes that are relevant to the layer, such as a name or a link to its spatial reference system. The `representations` relation associate, to each layer, its geometries relations. The model allows multiple spatial representations for the objects of the layer. The `attributesRel` relation points to the descriptive attributes relations associated to a layer.

This data model is suitable to store static layers. It allows more than one attribute relation for each object type, as needed by real data. Mapping the example shown in Figure 1 to this database model, and including some data for clarity, we have the following data relations:

DistrictsG

geomId: **int**	objectId: **string**	spatialData: **spatial**
1	261160605001	*bbbbbbbbbbbbbbbbb*
2	261160605010	*bbbbbbbbbbbbbbbbb*
3	261160605011	*bbbbbbbbbbbbbbbbb*

DistrictsA

ID_: **string**	name:**string**	POP_2004:**int**
261160605001	Recife	98361
261160605010	Cabanga	102548
261160605011	Sao Jose	1524

The metadata relations would contains the following items:

layers

layerId: **int**	layerName: **string**
1	*Recife*

representations

layerId: **int**	geomRelation: **string**
1	*DistrictsG*

attributesRel

layerId: **int**	attrRelation: **string**
1	*DistrictsA*

3.2.2 Temporal layers

In this section, we consider how to extend the static model to deal with temporal data. Suppose that we want to keep track of changes in each district, to follow the evolution of its population and its boundaries. We want to be able to register all of these changes in the same database and to extract information about the spatial and temporal changes on the data. Since we consider that an *ST-object* preserves its identity over time, every change in its attributes or its geometries produces a new instance of this object. Since changes in attributes and geometries might be asynchronous, our generic database model needs two adjustments. First, we have to introduce a unique identifier in every attribute relation. This identifier allows the distinction of different instances of attributes to the same object. The second adjustment is including one more relation, a *status* relation. This relation describes which instances of geometries and attributes are valid in a given interval:

status(geomId: **int**, uniqueId: **string**, initialTime: **time**, finalTime: **time**)

The status relation maps every instance of geometry (identified by the field geomId) to an instance of attributes values (identified by the field uniqueId). Each mapping has an associated valid interval (identified by the field initialTime and finalTime). As is possible that a layer has more than one attribute and geometry relations, there should be one status relation to each combination of a geometry relation with an attribute relation. Returning to our example of the districts of Recife, and showing some data, we would have the following data relations:

DistrictsG

geomId: **int**	objectId: **string**	spatialData: **spatial**
1	261160605001	*bbbbbbbbbbbbbbbbb*
2	261160605010	*bbbbbbbbbbbbbbbbb*
3	261160605011	*bbbbbbbbbbbbbbbbb*
4	261160605001	*bbbbbbbb*
5	261160605010	*bbb*

DistrictsA

ID_: **string**	name:**string**	POP_2004:**int**	unique_id: **string**
261160605001	Recife	98361	1
261160605010	Cabanga	102548	2
261160605011	Sao Jose	1524	3
261160605011	Sao Jose	2789	4
261160605010	Cabanga	106548	5

DistrictsStatus

attributeInst:**string**	geomInst:**string**	timeI:**time**	timeF: **time**
1	1	01/01/2003	31/12/2003
2	2	01/01/2003	31/12/2003
3	3	01/01/2003	31/12/2003
1	4	31/12/2003	
4	3	31/12/2003	
5	5	31/12/2003	

The status relation tracks both synchronous and asynchronous changes in the geometries or attribute values of the *ST-Objects*. It also allows the retrieval of individual *ST-Instances* of an *ST-Object*.

3.3. Examples of Spatio-Temporal Data

This section describes four sets of the spatio-temporal data that are representative of real world problems and demands. They are semantically different, and explain the requirements for the query processor and the expressive power of our generic database model. These four data sets are:

- *Crime events*: the object is the crime occurrence. Its geometry is a point representing the location of the event. Each new event has a unique identifier, therefore, is a new object. The main characteristic of this data set is that for each object there is only one spatio-temporal instance.
- *Traps to count mosquito eggs*: this data set originated from a joint work with public health researchers studying dengue fever in the Recife, Brazil. A set of traps were arbitrarily distributed along the city. Each trap attracts the mosquito female to lay her eggs in a special material inside the trap. Each trap has a unique identifier and once positioned its location does not change anymore. Researchers visit each trap weekly and count the number of eggs laid in the trap. The object is the trap, and the number of the eggs counted is the attribute that changes over time. Each new count creates a new instance of the object (the trap).
- *Land parcels*: this data is an experimental data that reflects the changes happening in land parcels of a neighborhood or city. Parcels are divided, sold or reacquired over time. This means that they can suffer changes in its geometry (for example, when the parcel is divided) or in their attributes (for example, when the parcel is sold). The object is the parcel, and each change in their attribute values or geometries generates a new instance of the parcel.
- *Satellite tracking animals*: this data results from a research project of surveying free ranging animals by radio transmitters. The transmitters are installed in collars attached to the animals and the signal is picked up by satellites. The object is the animal and each new signal detected is a new instance of the geometry of the object (or the animal location). This data represents the typical case of moving objects.

3.4. The Query Processor

A spatio-temporal database can be queried in different ways, according to the applications. For example, "*for each month, which changes occurred in the parcels?*", "*Which crimes happened on Friday in the south zone of Recife?*" or "*how many eggs were counted by trap in each month?*" *"What was the path followed by this wolf in July of 2004?"*. To answer this demand, we have developed a flexible query processor, named Querier, able to deal with different applications of spatio-temporal data and their needs for queries and responses. The Querier is responsible for the link between applications and spatio-temporal databases, as shown in Figure 2.

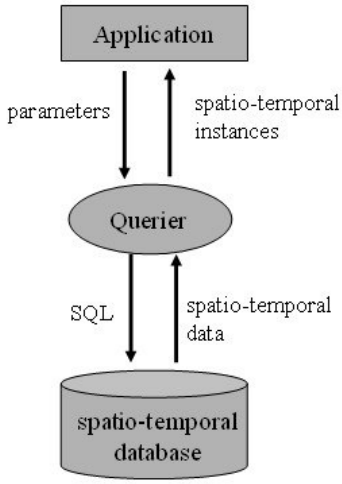

Figure 2. The Querier architecture.

The Querier receives a set of parameters that define its behavior. These parameters control how applications retrieve *ST-Instances* and *ST-Objects*. Their combination provides a flexible way of querying a spatio-temporal database:

- Layer: the source of the *ST-Instances* recovered by the Querier.
- Chronon: defines how the *ST-Instances* are split in time frames. The possible values of chronon are: `second`, `minute`, `hour`, `day`, `month`, `year`, `day of week`, `day of month`, `day of year`, `month of year`, `week of year`, `hour of day`, `minute of hour` or `second of minute`. For example, if you want to watch the crime events separated by weekday, the Querier returns seven time frames, one for each weekday. In this case, all the crimes that happened on Sunday are in the first time frame, the ones that happened on Monday are in the second time frame, and so on.
- Which time frames will be retrieved: controls whether the Querier returns either every time frame existent in the interval or only the time frames where some change occurred. Suppose a user is querying a set of parcels in 2004 and Chronon `month`. Changes on the parcels occurred only in January and in September of 2004. If the user requests the Querier to return every time frame, she will get twelve time frames. Otherwise, if she requests the Querier to return time frames where a change occurred, she will only get two time frames.
- Temporal predicate: the possible values for this parameter are the temporal interval predicates defined by Allen [1983]: `equals`, `before`, `after`, `meets`, `during`, `overlaps`, `ends` or `starts`.
- Aggregate functions: allows grouping a set of *ST-Instances* to produce a single value. The functions supported by the Querier are: `maximum`, `minimum`, `average`, `sum`, `counting`, or `variance`. This parameter should be used, for example, when one needs to know the average of eggs counted by trap is in each month.

- Spatial predicate: Based on Egenhofer [1994], the possible spatial predicates are: `disjoint`, `touches`, `crosses`, `within`, `overlaps`, `contains`, `intersects`, `equals`, `covers`, and `covered by`.

3.4.1 Querier Examples

In this section we show some examples of how to use the Querier to retrieve spatio-temporal data. We present the different ways of using the Querier combining the set of parameters to answer some usual questions.

Example 1) Returns the crime events from database, its location and its properties.

Step 1. Retrieve the layer information

```
CrimeLayer = database->getLayer ("Crimes");
```

Step 2. Set the parameters

```
Querier->setParams(loadGeometries, loadAllAttributes);
Querier->setParams(CrimeLayer);
```

Step 3. Load the instances

```
Querier->loadInstances();
```

Step 4. Consume the returned instance (ST-instances).

```
while (Querier->fetchInstance(sti))
{
     Geometry = sti.getGeometry();
     Properties = sti.getProperties();
     Time = sti.getTime();
}
```

In this example, the Querier returns the crimes from the layer "Crimes", its location (point geometry) and its attributes or properties. The first step loads the layer information from the database. The Querier behavior is defined by the parameters set in the step 2. Step 3 loads the instances and, in the final step, the Querier traverses every crime event and, for each one, gets its location (geometry), properties and time.

Example 2) Which crimes happened in 2003 in the district named "Cabanga" of Recife city?

Step 1. Retrieve the layer information

```
CrimeLayer = database->getLayer ("Crimes");
```

Step 2. Get the geometry from the layer that will be used in the spatial predicate

```
DistrictLayer = database->getLayer ("Districts");
DistrictGeometry= DistrictLayer->getGeometry ("Cabanga");
```

Step 3. Set the parameters

```
Querier->setParams(loadGeometries, loadAllAttributes);
Querier->setParams(CrimeLayer, year, ChangedTimeFrames);
```

```
Querier->setSpatialRestriction(DistrictGeometry, within);
```

Step 4. Load the instances of a specific time frame

```
TimeFrame = Querier->getTimeFrame('2003', Year);
Querier->loadInstances(TimeFrame);
```

Step 5. Consume the returned instance (ST-instances).

```
while (Querier->fetchInstance(sti))
{
    Geometry = sti.getGeometry();
    Properties = sti.getProperties();
    Time = sti.getTime();
}
```

In this example the Querier returns the crimes events from the layer "Crimes" that spatially occurred within the district boundaries named "Cabanga". The information about the layer that will be queried is loaded in step 1 and the district boundaries are retrieved from the layer "Districts" in step 2. Step 3 sets the Querier parameters and its spatial predicate. The crimes are grouped by years, since we defined the chronon as `year`. This Querier is set to return only the time frames with changes (`ChangedTimeFrames`). Thus, each returned instance has at least one crime. The spatial predicate is defined by the district boundaries and by the spatial relation `within`.

The Querier loaded only the instances of the time frame associated to year 2003, as shown in the step 4. In the last step, the Querier traverses every crime that happened in 2003 and within the district "Cabanga".

Example 3) How many eggs were counted in each trap in each month?

Step 1. Retrieve the layer information

```
TrapLayer = database->getLayer ("Traps");
```

Step 2. Define the aggregation function

```
GroupingAttributes->insert ("num_eggs", SUM);
```

Step 3. Set the parameters

```
Querier->setParams(loadGeometries, GroupingAttributes);
Querier->setParams(TrapLayer, month, ChangedTimeFrames);
```

Step 4. Pass over each time frame loading its instances

```
numTimeFrames = Querier->getNumTimeFrames();
for (frame=0 to numTimeFrames)
{
    Querier->loadInstances(frame);
```

Step 5. Consume the returned instance (ST-instances) for each time frame

```
    while (Querier->fetchInstance(sti))
```

```
        {
            Geometry = sti.getGeometry();
            Properties = sti.getProperties();
            Time = sti.getTime();
        }
    }
```

In this example, the Querier works with the layer "TrapLayer", grouping the egg counting by trap in each month (`month`). Thus, the Querier returns only one monthly *ST-Instance* for each trap. This instance is of the sum of the egg counts in a trap location. Step 2 defines that the attribute "num_eggs" will be aggregated by function `SUM`.

Example 4) Which changes occurred in the land parcels in each month?

Step 1. Retrieve the layer information

```
ParcelLayer = database->getLayer ("Parcels");
```

Step 2. Set the parameters

```
Querier->setParams(loadGeometries, loadAllAttributes);
Querier->setParams( ParcelLayer, month, ChangedTimeFrames,
                    Starts | Ends);
```

Step 3. Traverse each time frame loading its instances

```
numTimeFrames = Querier->getNumTimeFrames();
for (frame=0 to numTimeFrames)
{
    Querier->loadInstances(frame);
```

Step 4. Consume the returned instance (ST-instances) for each time frame

```
    while (Querier->fetchInstance(sti))
    {
        Geometry = sti.getGeometry();
        Properties = sti.getProperties();
        Time = sti.getTime();
    }
}
```

Each change in a parcel creates a new instance with an associated interval. This interval is composed of an initial time (when the validity of the instance started) and of a final time (when the validity of the instance ended). Thus, to recover which changes happened in a month, it is necessary to know which instances have an interval that started or ended in this month.

In this example, the Querier returns all the parcels that change in each month. Then, the temporal relation set in the Querier is the combination of the predicates *starts* and *ends* as shown in step 2. Finally, as in the previously examples, it traverses every time frames created by the Querier and retrieves the *ST-Instances* of each time frame, as shown in steps 3 and 4.

When the temporal relation between time frames and *ST-Instances* is not defined directly, as in the examples 2 and 3, the Querier returns all instances valid in each time frame. We say that an instance is valid in a time frame when its validity time has some intersection with the interval that represents the time frame.

4. A case study

This section shows a case study of the proposed query processor architecture. This case study uses the TerraLib library [Câmara, Souza et al. 2000]. TerraLib builds a layered database architecture on top of an existing object-relational DBMS (OR-DBMS) such as MySQL, PostgreSQL, PostGIS or ORACLE. TerraLib provides a database generic application interface that hides the differences among the spatial abstract data types provided by the OR-DBMS's. The generic database interface also hides differences between spatial data handling by these DBMS [Ferreira, Queiroz et al. 2002].

4.1. A Spatio-Temporal Database Observer

This section shows a graphical application with built using the TerraLib implementation of the query processor described in the sections above. The main use of the application is to show the status of the database, following it in different time frames. The user can choose to see all the time frames or only the time frames where a change occurred. The interface (shown in Figure 3) lets the user choose the layer and the query parameters. The right side of the interface allows to the user to group the instances, as shown in Example 3.

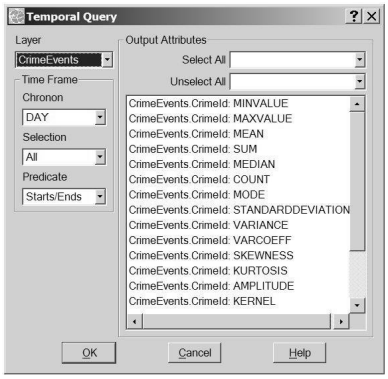

Figure 3. The selection interface.

4.1.1 Viewing crime events

Figures 4 and 5 show the first and the last time frames form the layer of crime events, using chronon `month`. The data in the database are the crime events from January 2000 to June 2003, with 42 time frames. As we move from one time frame to the next, the interface shows the matching crime events. Had we chosen the chronon Year, the interface would show only 4 time frames.

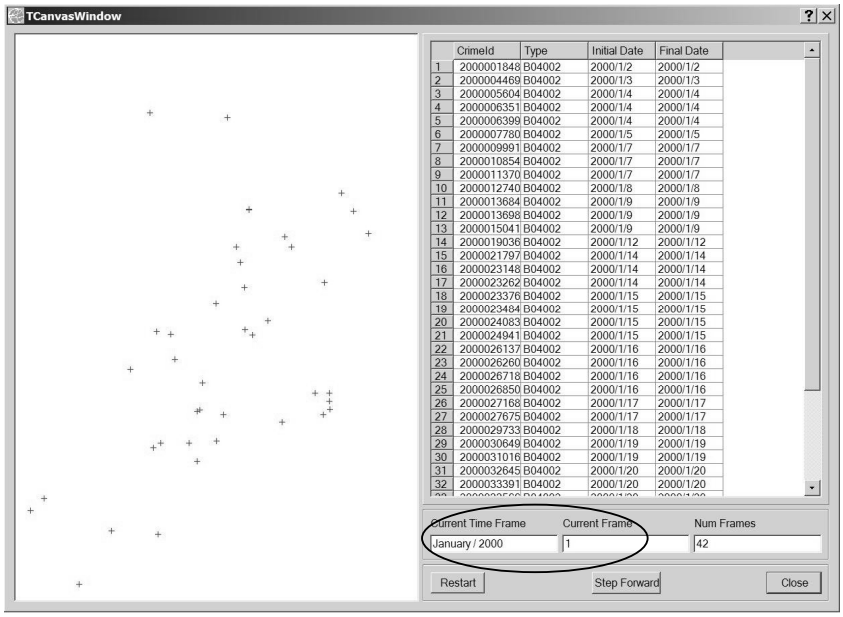

Figure 4. Watching the layer of crime events. The first time frame.

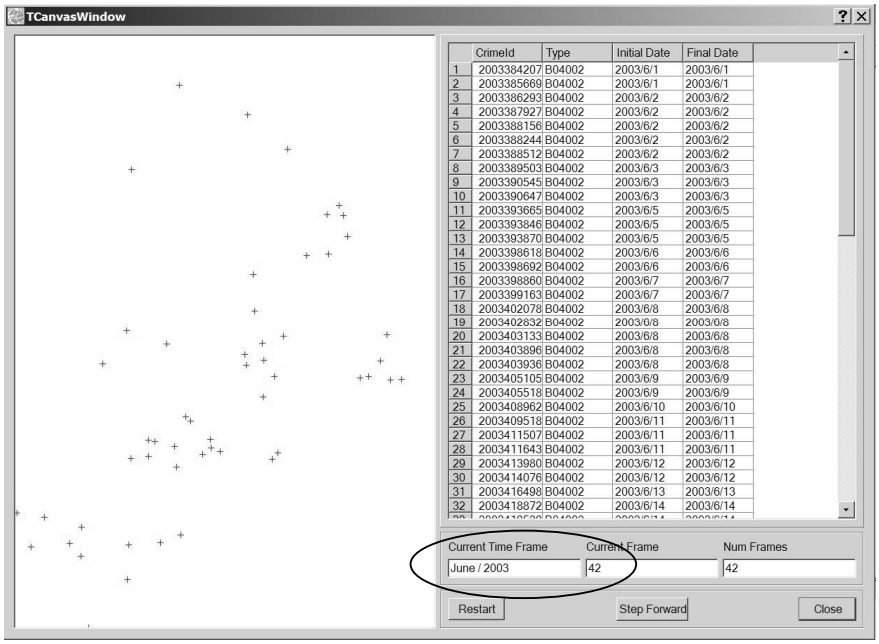

Figure 5. Watching the layer of crime events. The last time frame

4.1.2 Watching mosquito traps

If we follow the layer of traps and the eggs counts in each trap using chronon `day`, we can see the first and third frames have counts associated to the trap identified as *EM113*. Figures 6 and 7 show these two time frames.

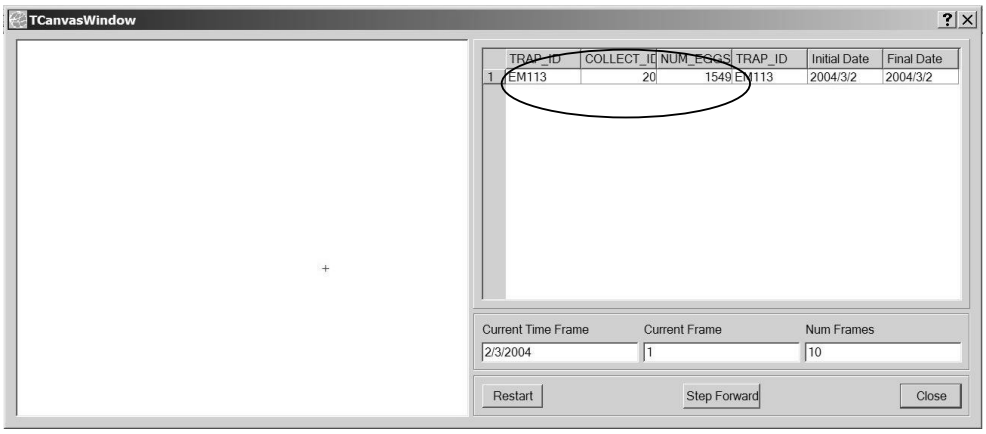

Figure 6. Watching the layer of traps. First time frame of chronon day.

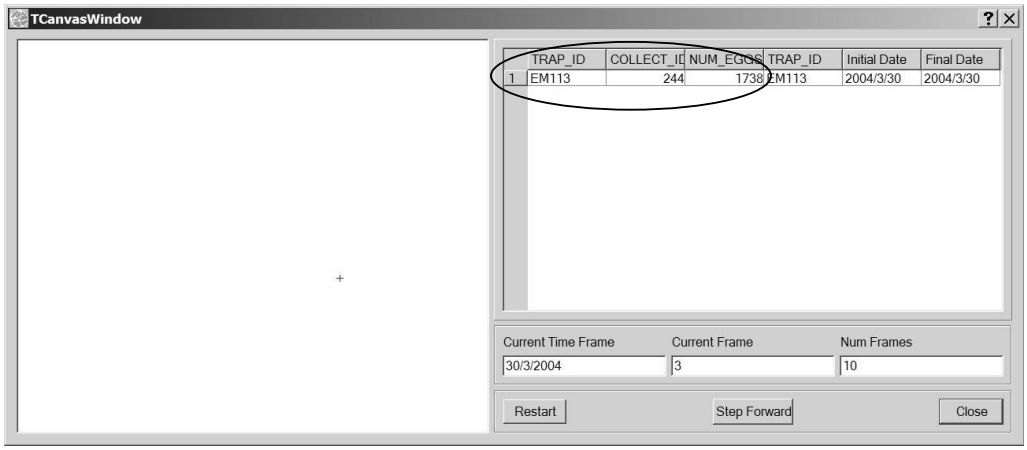

Figure 7. Watching the layer of traps. Third time frame of chronon day.

If we watch the layer of traps using the chronon Month, we will get more than one instance associated to the trap *EM113*, in the time frame March, 2004. This happens because there were two data collections in this trap during this month. Figure 8 shows the first time frame resulting from asking for the sum of eggs counted per month for each trap. The number of eggs in trap *EM113* is the sum of the number of eggs that appeared in the instances shown in Figures 6 and 7.

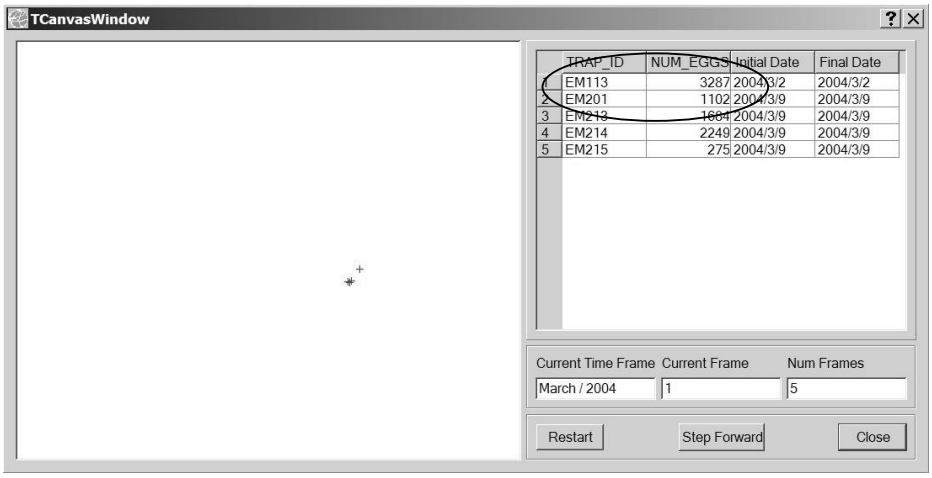

Figure 8. Watching the layer of traps. First time frame of chronon Month.

In the example, the grouping function is a sum over a descriptive attributes, using statistical measures. This grouping operation transforms a set of instances into a single one. As the mosquito traps do not change their location, this function makes sense in the application domain. In cases where the geometry of the instances of the same object are different, we would have to provide a similar way to choose what geometry is representative of all instances on an object in a time frame. This could be done, for example, choosing their intersection or union. We have not yet addressed this requirement, but the query processor is flexible enough to include this new parameter.

4.1.3 Watching change in parcels

The layer of parcels is more interesting, because it allows following changes in the geometries of the *ST-Objects*. Using chronon `Month`, and specifying that we want to see all the time frames available, there are 15 time frames. Figure 9 shows the first.

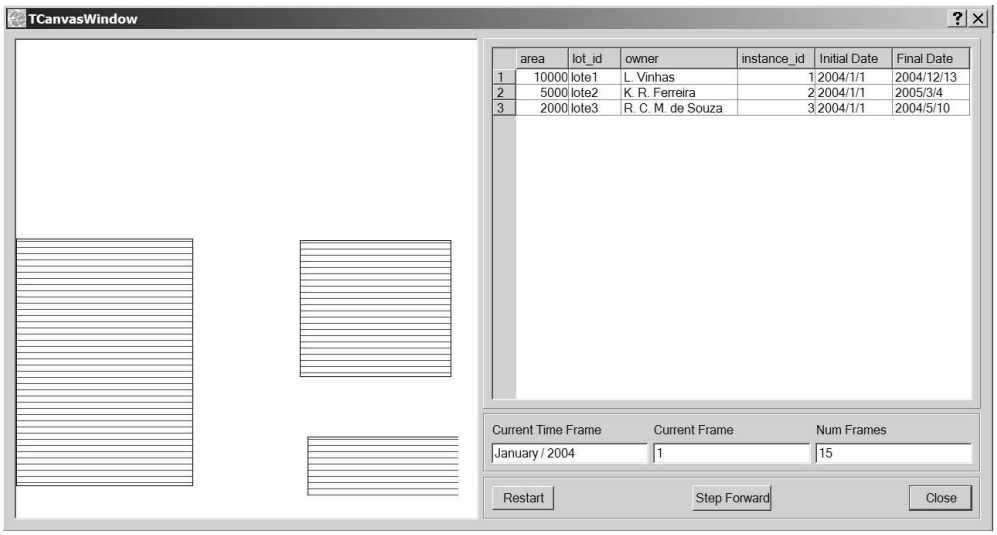

Figure 9. Watching the layer of parcels. First time frame of chronon Month.

Figure 10 shows the fiftieth time frame. We see that changes occurred in the attribute values of object *lote3* as shown in rows 1 and 2 of the table. We get two instances of this object. The geometry of the first instance of an object in the time frame is shown in horizontal pattern brush style, the second in a vertical pattern brush style. Looking at the resulting cross pattern we see the geometries of the two instances coincide, or there where no changes in the geometries of the object. Row 1 of the table shows that the attributes changed in 2004/5/10 and row 2 shows that the next instance is valid until present time.

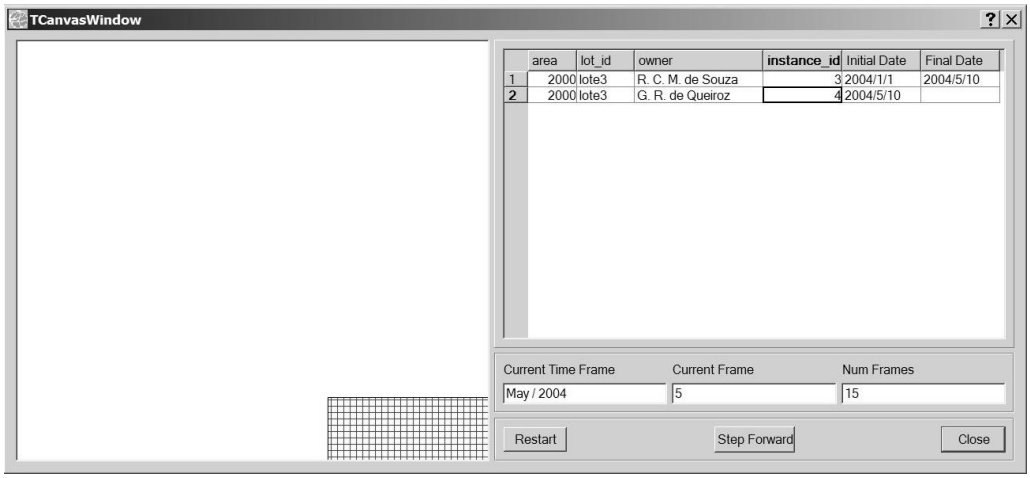

Figure 10. The fiftieth time frame of chronon Month, layer of parcels.

The Figure 11 shows the third change that happened in the twelfth time frame. The object *lote1* was divided, given origin to the object *lote4*. We note in the table that two instances of *lote1* where returned (rows 1 and 2). The cross pattern and double horizontal pattern shows the piece of *lote1* that was sold and created *lote4*.

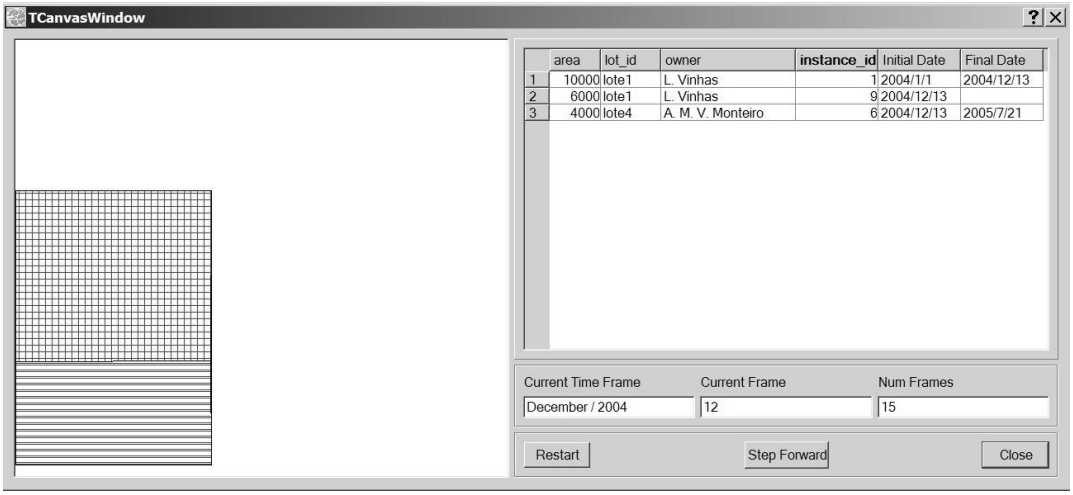

Figure 11. The twelfth time frame of chronon Month, layer of parcels.

A similar change occurred in object *lote2* as shows Figure 12.

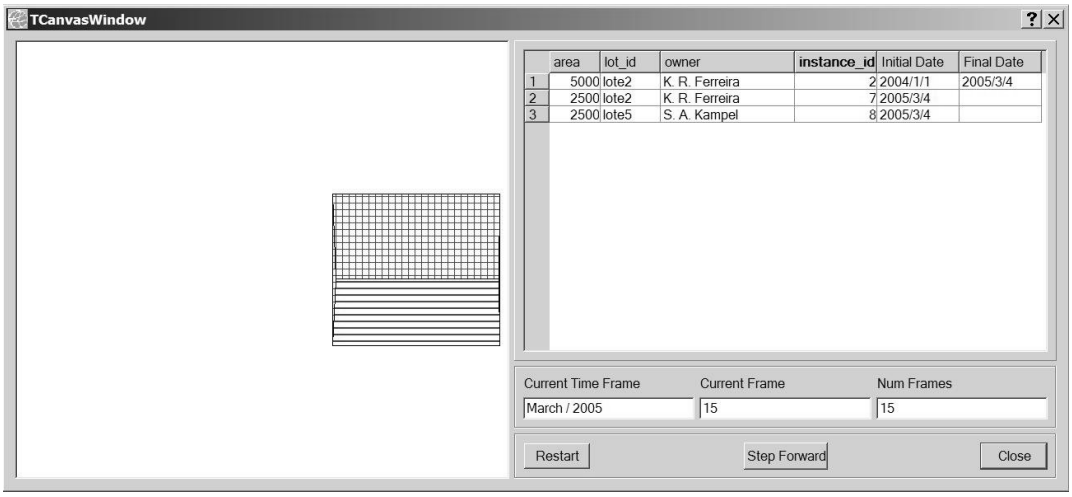

Figure 12. The fifteenth time frame of chronon Month, layer of parcels.

5. Conclusions

In this work, we presented a generic database model to deal with spatio-temporal data. Our idea is that the internal architecture of the database can be unified for different spatio-temporal applications. Using this data model, we built a flexible query processor that can answer the demands of different applications. This processor is defined through parameters that define the different combinations of spatial and temporal restrictions. We have shown examples of how the Querier follows the state of the database in different time frames. This functionality is useful, for example, to build spatio-temporal statistical functions. We have also shown how to use Querier to answer questions that relate the spatial and temporal aspects of the data.

We have implemented the Querier using the TerraLib library and its generic database interface. This implementation of the Querier allows the use of different DBMS to build the generic model of a spatio-temporal database. We validated this implementation in different application of spatio-temporal data. For future research we intent to couple the Querier parameters to a spatio-temporal language, so that it will be a spatio-temporal language processor. We also intend to build indexing structures to optimize query processing.

References

Allen, J. F. (1983). "Maintaining Knowledge about Temporal Intervals." Communications of the ACM **26**(11): 832-843.

Câmara, G., R. Souza, B. Pedrosa, et al. (2000). TerraLib: Technology in Support of GIS Innovation. II Brazilian Symposium on Geoinformatics, GeoInfo2000, São Paulo.

Clementini, E. and P. Di Felice (1996). "A Model for Representing Topological Relationships Between Complex Geometric Features in Spatial Databases." Information Sciences **90**(1-4): 121-136.

Couclelis, H. (1992). People Manipulate Objects (but Cultivate Fields): Beyond the Raster-Vector Debate in GIS. Theories and Methods of Spatio-Temporal Reasoning in Geographic Space. A. Frank, I. Campari and U. Formentini. Berlin, Springer-Verlag. **639:** 65-77.

Dieker, S. and R. H. Güting (2000). Plug and Play with Query Algebras: SECONDO, A Generic DBMS Development Environmen. Proc. of the Int. Database Engineering and Applications Symp. (IDEAS 2000).

Egenhofer, M. (1994). "Spatial SQL: A Query and Presentation Language." IEEE Transactions on Knowledge and Data Engineering **6**(1): 86-95.

Egenhofer, M. (1999). Spatial Information Appliances: A Next Generation of Geographic Information Systems. First Brazilian Workshop on GeoInformatics, Campinas, Brazil.

Egenhofer, M. and R. Franzosa (1991). "Point-Set Topological Spatial Relations." International Journal of Geographical Information Systems **5**(2): 161-174.

Erwig, M., R. H. Güting, M. Schneider, et al. (1999). "Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases." GeoInformatica **3**(3): 269-296.

Erwig, M. and M. Schneider (2002). "Spatio-Temporal Predicates." IEEE Transactions on Knowledge and Data Engineering **14**(4): 881-901.

Ferreira, K. R., G. Queiroz, J. A. Paiva, et al. (2002). Arquitetura de Software para Construção de Bancos de Dados Geográficos com SGBD Objeto-Relacionais. XVII Simpósio Brasileiro de Banco de Dados, Gramado, RS.

Frank, A. U. (2003). Ontology for spatio-temporal databases. Spatiotemporal Databases: The Chorochronos Approach. T. Sellis. Berlin, Springer-Verlag.

Güting, R. H., M. H. Bohlen, M. Erwig, et al. (2003). Spatio-temporal Models and Languages: An Approach Based on Data Types. Spatio-Temporal Databases. M. Koubarakis. Berlin, Springer.

Güting, R. H. and M. Schneider (2005). Moving Objects Databases. New York, Morgan Kaufmann.

Hornsby, K. and M. Egenhofer (2000). "Identity-Based Change: A Foundation for Spatio-Temporal Knowledge Representation." International Journal of Geographical Information Science **14**(3): 207-224.

Medak, D. (2001). Lifestyles. Life and Motion of Socio-Economic Units. ESF Series. A. U. Frank, Raper, J., & Cheylan, J.-P. London, Taylor & Francis.

Papadias, D. and M. Egenhofer (1997). "Hierarchical Spatial Reasoning about Direction Relations." GeoInformatica **1**(3): 251-273.

Parent, C., S. Spaccapietra and E. Zimányi (1999). Spatio-temporal conceptual models: data structures + space + time. 7th ACM international symposium on Advances in geographic information systems, Kansas City, USA, ACM Press.

Pelekis, N., B. Theodoulidis, I. Kopanakis, et al. (2004). "Literature review of spatio-temporal database models." The Knowledge Engineering Review **19**(3): 235 - 274

Relly, L., H.-J. Schek, O. Henricsson, et al. (1997). Physical Database Design for Raster Images in Concert. 5th International Symposium on Spatial Databases (SSD'97), Berlin, Germany, Springer.

Tryfona, N. and C. Jensen (1999). "Conceptual Data Modeling for Spatiotemporal Applications." GeoInformatica **3**: 245-268.

Worboys, M. (1994). "A Unified Model for Spatial and Temporal Information." The Computer Journal **37**(1): 27-34.