Design of Spatio-Temporal Query Languages
– Workshop Position Paper –

Martin Erwig
Oregon State University
erwig@cs.orst.edu

1 Spatial Data as Abstract Data Types

In contrast to the *field view* of spatial data that basically views spatial data as a mapping from points into some features, the *object view* clusters points by features and their values into spatial objects of type *point*, *line*, or *region*. When embedding these objects into a data model, such as the relational model, an additional clustering according to conceptually identified objects takes place. For example, we could define a relation

$$City(name : string, center : point, area : region)$$

that combines different features for cities in one relation. An important aspect of this kind of modeling is that clustering happens on two different levels: (i) points are grouped into spatial objects like regions and (ii) different attributes/features are grouped into a perceived object.

When talking about data modeling there is no reason why this grouping should be limited to two levels. For example, we can consider storing regions of different population densities for each city in an attribute $density : num \to region$. Although then the relation is not in first normal form anymore, we can "recover" the first normal form by encapsulating the function $num \to region$ in an abstract data type. The important aspect is that all the required operations on such a type as well as on regions and other complex types can be defined to a large degree independently from the data model.[1]

The most important point about the preceding discussion is the way in which complex types can be easily integrated into a data model, namely as *abstract data types* (ADTs); they are *not* modeled by abstractions offered by the data model. An important benefit of this approach is that the design and definition of operations that can be used in query languages is not ruled by (implementation aspects of) the data model, but is driven from the application of the objects/types under consideration.

2 An Abstract Data Type for Spatio-Temporal Data

We can apply the ADT idea to model spatio-temporal data and to integrate such ADTs into data models. The basic idea is very simple and starts from the observation that anything that changes over time can be regarded as a function that maps from time into the data of consideration [1]. For example, the movement of a satellite (projected onto the earth's surface) can be regarded as a function $Sat : time \to point$ or the development of an oil spill can be regarded as a function $Spill : time \to region$; see Figure 1. We can apply functions like *Sat* or *Spill* to time values and obtain the position of the satellite or the extent of the oil spill at a particular time as spatial data.

Of course, *spatial* temporal data is just a particular example of temporal data in general. For example, we can also deal with temporally changing numbers through a type like $time \to num$. Such a temporal number could give the size of the oil spill depending on the time. Using the notational convention that temporal versions of data types have their letter capitalized, we have $Sat : Point$ and $Spill : Region$ and we can define functions on spatio-temporal objects, such as:

[1] However, operations that compute collections of objects must probably return them in a form that is compatible with the data model, for example, an intersection operation on regions could return a relation of intersection results.

$$Area : Region \rightarrow Num$$

If we apply *Area* to *Spill*, we obtain a time-dependent number reflecting the temporal development of the size of the oil spill. Many operations like this are categorized and described in [6].

By encapsulating (spatio-) temporal data in data types like *Point*, *Region*, and *Num* we can easily integrate it into data models (at least on the conceptual level). Taking the relational model again, we can define two relations containing information about satellites and pollution incidents:

$$Satelites(sname : string, Pos : Point)$$
$$Pollutions(pname : string, Reg : Region)$$

Notice how the temporal changes are completely encapsulated by the ADTs *Point* and *Region*.

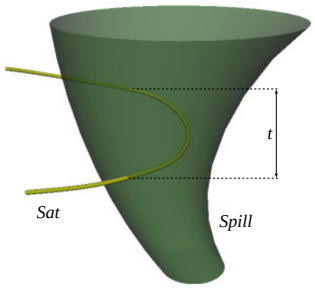


Figure 1: Satellite movement and oil spill development.

3 Spatio-Temporal Query Languages

The integration of spatio-temporal data types into data models supports the definition of a query languages because the interaction between ADT operations and general querying constructs is constrained to a few well-defined places. We are immediatley able to pose spatio-temporal queries in a straightforward way. For example, we can ask which satellites were able to observe which pollution incidents for a certain amount of time by (note that $hours : num \rightarrow time$):

SELECT *sname*, *pname* FROM *Satellites*, *Pollutions*
WHERE $time(Intersection(Pos, Reg)) > hours(3)$

Here, *Intersection* is an ADT function that computes (in the case of a *Point* and a *Region* argument) a *Point* value that represents that parts of the sattelites' route that are inside a pollution area. The function *time* computes the domain of temporal functions and gives in this case the time interval(s) of the qualifying route parts.

Although many interesting queries can be expressed by this approach, there are also many queries that cannot be formulated. One large class of such queries are queries asking for changes in topological relationships. For example, we might be interested in finding ships that were inside the oil spill and eventually left the spill. Using a spatio-temporal intersection predicate yields also all ships that entered, crossed, touched, etc. the spill and is not precise enough to find the desired information. With the introduction of *spatio-temporal predicates* [4] we can, however, describe very precisely developments of objects and their relationships. Spatio-temporal predicates can be thought of as predicates that are built by regular expressions over a basic set of spatial and spatio-temporal predicates. For example, the predicate describing a "leaving" development can de defined as:

$$Leaves := Inside\, meet\, Disjoint$$

so that we can pose a corresponding query [2, 5] (we assume here that *Ships* is relation a with the same attributes as *Satellites*):

SELECT *sname* FROM *Ships*, *Pollutions*
WHERE *pname* = *'That Big Spill'* AND *Pos Leaves Reg*

Another class of queries that cannot be expressed with our current set of ADT operations is actually characterized be the provided example query "Where did spring arrive earlier this year than last year?" Assuming that we represent spring as an evolving region $Spring : Region$, a difference operator on regions lifted to the spatio-temporal level can only reveal where and when spring *was* this year and not last year, but it cannot tell where it *arrived* earlier. What we need is actually a derivative operator *Derive* that can compute the changes of spatio-temporal objects. The derivative of a *Region* would be, in general, a function from time

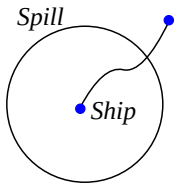
into a domain containing signed curves and regions (with certain constraints, for example, regions can occur only for isolated time points). If, as the example suggests, spring data is given in discrete form, the result will be just regions. To shorten the discussion, we further assume that we have two operators *DerivePos* and *DeriveNeg* that return proper *Region* objects for positive and negative derivatives, respectively. Then the arrival of spring is given by the expression *DerivePos*(*Spring*). To express the "earlier" constraint we can employ a generalized projection operator *Project* that projects a spatio-temporal value, such as *Region*, to a set of corresponding spatial values paired with a list of the time points from which they were mapped. Now the query can then be expressed as:

$$Project(DerivePos(Spring)) \textbf{ where } \lambda(r, [t, t']).t - t' < year(1)$$

4 A(E)nd Users?

Having defined powerful query facilities, we might ask ourselves who else will be ever able to use them? In particular, non-computer scientists do not want to learn new languages (programming languages, query languages, or others); they just want to get their job done, which means that end users need an easy access to spatio-temporal data and queries.

One possible answer to this problem is to implement a couple of queries and offer these hardwired at a tailor-made user interface. This strategy, however, means a severe access restriction for end users.



Another approach is to define visual query languages that allow the user to express arbitrary queries without having to master the syntax of a rigid textual query language. For example, we have defined a visual language and query interface that allows users to draw sketches of object traces, which are then automatically translated into queries using spatio-temporal predicates [3]. A sketch for the predicate *Leave* is shown on the right.

5 Position Statement

I believe that the ADT or functional modeling approach helps to structure the comprehension of spatio-temporal data. In particular, the definition of types and operations can serve as interfaces between the different levels of a spatio-temporal data model and its implementation. With types and operations we can express requirements on lower levels and offer services to higher levels. A layered approach with clearly defined interfaces helps to modularize the development of spatio-temporal database systems and applications and supports collaborative efforts of different research groups.

References

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

[2] M. Erwig and M. Schneider. Developments in Spatio-Temporal Query Languages. In *IEEE Int. Workshop on Spatio-Temporal Data Models and Languages*, pages 441–449, 1999.

[3] M. Erwig and M. Schneider. Query-By-Trace: Visual Predicate Specification in Spatio-Temporal Databases. In H. Arisawa and T. Catarci, editors, *Advances in Visual Information Management – Visual Database Systems*, pages 199–218. Kluwer Academic Publishers, Boston, MA, 2000.

[4] M. Erwig and M. Schneider. Spatio-Temporal Predicates. *IEEE Transactions on Knowledge and Data Engineering*, 2002. To appear.

[5] M. Erwig and M. Schneider. STQL: A Spatio-Temporal Query Language. In R. Ladner, K. Shaw, and M. Abdelguerfi, editors, *Mining Spatio-Temporal Information Systems*. Kluwer, 2002. To appear.

[6] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. *ACM Transactions on Database Systems*, 25(1):1–42, 2000.