Chapter 1

STQL — A SPATIO-TEMPORAL QUERY LANGUAGE

Martin Erwig
Oregon State University
Department of Computer Science
Corvallis, OR 97331, USA
erwig@cs.orst.edu

Markus Schneider
University of Florida
Department of Computer and Information Science and Engineering
Gainesville, FL 32611, USA
mschneid@cise.ufl.edu

Abstract Integrating spatio-temporal data as abstract data types into already existing data models is a promising approach to creating spatio-temporal query languages. Based on a formal foundation presented elsewhere, we present the main aspects of an SQL-like, spatio-temporal query language, called STQL. As one of its essential features, STQL allows to query and to retrieve *moving objects* which describe *continuous* evolutions of spatial objects over time. We consider spatio-temporal operations that are particularly useful in formulating queries, such as the temporal lifting of spatial operations, the projection into space and time, selection, and aggregation. Another important class of queries is concerned with *developments*, which are changes of spatial relationships over time. Based on the notion of *spatio-temporal predicates* we provide a framework in STQL that allows a user to build more and more complex predicates starting with a small set of elementary ones. We also describe a *visual notation* to express developments.

Keywords: Moving object, spatio-temporal predicate, development, projection, selection, aggregation, abstract data type

1. Introduction

Motivated by the deep relationships between temporal and spatial phenomena there has recently been an increased interest in designing *spatio-temporal data models* and *spatio-temporal databases* that deal with geometries changing over time. Our main objective in this area is to provide a DBMS data model and query language capable of handling such time-dependent geometries. Thereby, we especially focus on geometries changing *continuously* over time, and we call them *moving objects*. Our modeling also includes the simpler, discrete case where temporal evolutions are stepwise constant. Two fundamental abstractions are *moving points* (like cars, planes, animals, people) and *moving regions*, or better: *evolving regions* (like storms, people, temperature zones, high/low presure areas), which describe objects for which only the time-dependent position, or position and extent, respectively, are of interest. We propose to represent such time-dependent geometries as attribute data types with suitable operations and predicates. In other words, our approach is based on defining an abstract data type extension to a DBMS data model and query language. The main benefit of this approach is that our type system is in principle independent of a specific DBMS data model and can be embedded into any query language, like a relational, object-relational, or object-oriented language.

Based on a formal foundation published in earlier papers (see next section), we introduce in this chapter the essential features of STQL as a spatio-temporal query language for moving objects. We have placed STQL into a relational setting as an extension of the well known standard relational query language SQL. We demonstrate how an integration of spatio-temporal operations and predicates into SQL can be performed. This integration profits very much from the abstract data type approach for integrating complex objects into databases. The spatio-temporal operations that we consider include those that are obtained by temporal lifting of spatial operations, projections into space and time, selection, and aggregation. A special concern relates to *spatio-temporal predicates* as components of so-called *developments* which describe temporally changing spatial relationships of moving objects. Finally, we demonstrate how the user queries employing spatio-temporal predicates can be supported by a visual query language interface.

The rest of this chapter is structured as follows. In Section 2 we discuss related work. Section 3 describes the underlying data model and presents the main foundations for spatio-temporal operations, predicates, and developments. In Section 4 we present the essential features of STQL by query examples. Section 5 sketches how visual languages and query interfaces can be employed for a simple, user-friendly formulation of powerful queries. Finally, Section 6 draws some conclusions.

2. Related Work

So far, a few data models for spatio-temporal data have been proposed. In [Worboys, 1994] a spatial data model has been generalized to become spatio-temporal. Spatio-temporal objects are defined as so-called spatio-bitemporal complexes whose spatial features are described by simplicial complexes and whose temporal features are given by bitemporal elements attached to all components of simplicial complexes. On the other hand, temporal data models have been generalized to become spatio-temporal and include variants of Gadia's temporal model [Gadia and Nair, 1993] which can be found in [Cheng and Gadia, 1994; Böhlen et al., 1998]. The main drawback of all these approaches is that they are incapable of modeling *continuous* changes of spatial objects over time.

The constraint-based approach to modeling spatio-temporal data, which is pursued, for example, in [Grumbach et al., 1998], considers spatio-temporal objects as point sets in a multi-dimensional space. The description of objects is given by logical formulas (constraints). Moreover, queries are also expressed by logical formulas. Although the logical/constraint-based approach is very general, it is not clear, for example, whether it can be efficiently implemented. Other possible problems are the handling of objects that require non-linear constraints and the treatment of metric operations (for example, computing distances).

Our approach, which will be the basis of this chapter, introduces the concept of *spatio-temporal data types* [Erwig et al., 1998b; Erwig et al., 1998a; Erwig et al., 1999; Güting et al., 2000]. The definition of a temporal object in general is based on the observation that anything that changes over time can be expressed as a function over time. A temporal version of an object of type α is then given by a function from *time* to α. Spatio-temporal objects like *moving points* and *evolving regions* are regarded as special instances of temporal objects where α is a spatial data type like for *points* and *regions* [Güting and Schneider, 1995].

Similar to the approach just mentioned, in [Yeh and Cambray, 1993; Yeh and Cambray, 1995] based on the work in [Segev and Shoshani, 1993] *behavioral time sequences* are introduced. Each element of such a sequence contains a geometric value, a date, and a *behavioral function*, the latter describing the evolution up to the next element of the sequence. Time sequences could be used as representations for our temporal objects.

An issue that has been intensively discussed in temporal data modeling is whether a tuple-timestamped or an attribute-timestamped data model should be preferred. Tuple-timestamped models (for example, [Ariav, 1986; Clifford et al., 1993; Snodgrass, 1993] expand the schema of a relation by one or more explicit temporal attributes that are used for describing the lifespan or validity period of a whole tuple. Each time an attribute of a tuple changes its value, the tuple has to be duplicated and modified. Hence, the information about an object is scattered over one or more relations. This approach impedes the modeling of continuous changes of spatial objects and the definition of corresponding operations and predicates; thus they are inappropriate for our purposes.

Instead of adding additional attributes to the relation schema, attribute-time-stamped models (for example, [Clifford and Croker, 1993; Clifford et al., 1993; Gadia and Nair, 1993; Segev and Shoshani, 1993]) aim at gathering information about an object into a single tuple and allow complex attribute values. These complex values incorporate the temporal dimension and are frequently modeled as functions from time into a value domain. From this perspective, attribute-timestamped models are very similar to and fit very well with our model.

However, our model additionally encapsulates (spatio-) temporal objects as ADT objects that can be integrated as complex values into databases [Stonebraker et al., 1983; Stonebraker, 1986]. The ADT approach has several advantages. The first very important benefit is that employing ADTs is more expressive than relying on attribute-timestamped models, let alone tuple-timestamped models, since continuous changes can be modeled [Erwig et al., 1999]. The second benefit is that a definition of ADT values is valid regardless of a particular DBMS data model and query language. The reason is that ADT values are *not* modeled by concepts of a DBMS data model and that they therefore do not depend on them. This facilitates an easy and elegant conceptual integration of ADT values into relational, complex-object, object-oriented, or other data models and query languages.

The query facility of SQL is provided by the well known `SELECT-FROM-WHERE` clause. The integration of predicates like "<" or "<>" for standard data types such as integers or strings is well understood. In particular, there are only a few of them which allows one to include them as built-in predicates. When considering more complex and more structured data such as points, lines, or regions, one can try to systematically derive all reasonable predicates. The so-called 9-intersection model [Egenhofer and Franzosa, 1991; Egenhofer and Herring, 1990] provides such canonical collections of predicates for each combination of spatial data types. For example, for two regions the eight predicates *disjoint*, *meet*, *overlap*, *coveredBy*, *covers*, *inside*, *contains*, and *equal* have been identified. A spatial query language based on these predicates and called *Spatial SQL* has been proposed in [Egenhofer, 1994].

From an application point of view, we have found that expressing and querying temporal changes or *developments* of spatial objects is an important feature of a spatio-temporal query language. For this purpose *spatio-temporal predicates* are needed, which model these developments and which can be used in the query part of STQL. A spatio-temporal predicate makes statements about the validity of the behavior of two spatio-temporal objects for some period.

In [Erwig and Schneider, 2002] we have introduced the concept of spatio-temporal predicates as functions mapping spatio-temporal objects into booleans. Moreover, we have described a generic way to construct spatio-temporal predicates from spatial predicates by lifting and temporal aggregation. For example, compute a binary spatial predicate for all the spatial values that are obtained from two spatio-temporal objects along the time axis and aggregate the resulting temporal boolean. The problem is tackled on the basis of the 9-intersection model and on the basis of the work in [Egenhofer and Al-Taha, 1992] which considers *possible topological transitions* (that is, changes) of topological relationships.

Between a moving point and an evolving region we have identified a canonical collection of 28 spatio-temporal predicates, and between two evolving regions we have obtained not less than 2198 predicates [Erwig and Schneider, 2002]. The large numbers practically impede a naming of all these predicates and their reasonable employment from a user perspective. A first solution to this problem could be to furnish the user with a small, fixed, application-specific set of predicates, which might be too restrictive. An alternative could be to pursue a strategy like in [Clementini et al., 1993]. There, an extension of the 9-intersection model by additionally taking the dimension of the intersections into account leads to 52 possible relationships for all combinations of point, line, and region objects. The large number of predicates is reduced by grouping all topological cases into five overloaded topological predicates and by providing two boundary operators. These five predicates are mutually exclusive and capture all possible topological relationships. In our case, the number of predicates is much larger. Moreover, new predicates can be constructed from already existing ones. Hence, we advocate an extensible approach and provide a simple framework for composing spatio-temporal predicates. The integration into SQL becomes possible by an appropriate *macro mechanism*. This is similar to the way in which composite events are specified in [Motakis and Zaniolo, 1995]. The main difference is that events occur always at some instant in time whereas we also deal with predicates over whole time periods.

3. The Data Model

For illustration purposes, we confine ourselves here to the well known relational model and to SQL as its most popular query language. A *relation scheme* R is written as $R(A_1 : D_1, \ldots, A_n : D_n)$ where the A_i are the *attributes* and the D_i are their respective value domains. For a relation $r : R(A_1 : D_1, \ldots, A_n : D_n)$ holds $r \subseteq D_1 \times D_2 \times \ldots \times D_n$. The domains can be standard types like integers, reals, booleans, or strings but also more complex types encapsulated into ADTs and including a comprehensive set of operations and predicates. Examples are spatial data types like points, lines, and regions [Güting and Schneider, 1995] or graphs [Erwig and Güting, 1994].

3.1 Moving Objects

Similarly, we model spatio-temporal data as abstract data types which can be employed as attribute types in a relation. The relation itself has only a container function to store attribute data in tuples. The design of the model for spatio-temporal data is as follows: for compatibility with smoothly changing spatio-temporal objects we choose a continuous model of time, that is, $\mathit{time} = \mathbb{R}$. The temporal version of a value of type α that changes over time can be modeled as a *temporal function* of type $\tau(\alpha) = \mathit{time} \rightarrow \alpha$. We have used temporal functions as the basis of an algebraic data model for *spatio-temporal data types* [Erwig et al., 1998b; Erwig et al., 1998a] where α is assigned a spatial data type like *point* or *region*. For example, a point that changes its location over time is an element of type $\tau(\mathit{point})$ and is called a *moving point*. Similarly, an element of type $\tau(\mathit{region})$ is a region that can move and/or grow/shrink. It is called an *evolving region*[1]. In addition, we also have changing numbers and booleans, which are essential when defining operations on temporal objects.

3.2 Temporal Lifting

To make notations more comprehensible we generally denote non-temporal types, entities, functions, and predicates by lower case letters while their temporal counterparts start with capital letters. For example, the spatial operation *distance* takes objects of type *point* and *region* and computes a number of type *real*, whereas its lifted version $\mathit{Distance} = \uparrow\!\mathit{distance}$ maps elements of type $\mathit{Region} = \tau(\mathit{region})$ and $\mathit{Point} = \tau(\mathit{point})$ to $\mathit{Real} = \tau(\mathit{real})$ (temporal reals). For instance, we could be interested in computing the (time-dependent) distance of an airplane and a storm. This could be achieved by an operation:

$$\mathit{Distance} : \tau(\mathit{point}) \times \tau(\mathit{region}) \rightarrow \tau(\mathit{real})$$

In principle, we can take almost any non-temporal operation and “lift” it so that it works on temporal objects returning also a temporal object as a result. More precisely, for each function $f : \alpha_1 \times \ldots \times \alpha_n \rightarrow \beta$ its corresponding lifted version is defined by [Güting et al., 2000]:

$$\uparrow\! f : \tau(\alpha_1) \times \ldots \times \tau(\alpha_n) \rightarrow \tau(\beta)$$

[1]Currently, we do not consider a temporal version of lines, mainly because there seem to be not many applications of moving lines. A reason might be that lines are themselves abstractions or projections of movements and thus not the primary entities whose movements should be considered [Erwig et al., 1999]. In any case, however, it is principally possible to integrate moving lines in much the same way as moving points or moving regions if needed.

with

$$\uparrow f(S_1, \ldots, S_n) \ := \ \{(t, f(S_1(t), \ldots, S_n(t))) \mid t \in \mathit{time}\}$$

For example, we have *Distance* $=$ $\uparrow$*distance*. Note that this definition implies lifting also for constant objects of a non-temporal type α, that is, $\uparrow : \alpha \rightarrow \tau(\alpha)$ with $\uparrow c \ := \ \{(t, c) \mid t \in \mathit{time}\}$. Temporal lifting is, of course, also applicable to spatial predicates. Consider the spatial predicate

$$\mathit{inside} : \mathit{point} \times \mathit{region} \rightarrow \mathit{bool}$$

The lifted version of this predicate has the type

$$\uparrow\mathit{inside} : \mathit{Point} \times \mathit{Region} \rightarrow \mathit{Bool}$$

with the meaning that it yields *true* for each time at which the point is inside the region, *undefined* whenever the point or the region is undefined, and *false* in all other cases.

3.3 Spatio-Temporal Predicates and Developments

A spatio-temporal predicate is essentially a function that aggregates the values of a spatial predicate as it evolves over time. In other words, a spatio-temporal predicate can be thought of as a lifted spatial predicate yielding a temporal boolean, which is aggregated by determining whether that temporal boolean was sometimes or always true. Thus, a spatio-temporal predicate is a function of type $\tau(\alpha) \times \tau(\beta) \rightarrow \mathbb{B}$ for $\alpha, \beta \in \{\mathit{point}, \mathit{region}\}$.

Consider again the definition of $\uparrow$*inside*. We can define two spatio-temporal predicates *sometimes-inside* and *always-inside* that yield true if $\uparrow$*inside* yields true at some time, respectively, at all times. Whereas the definition for *sometimes-inside* is certainly reasonable, the definition for *always-inside* is questionable, since it yields false whenever the point or the region is undefined. This is not what we would expect. For example, when the moving point has a shorter lifetime than the evolving region but is always inside the region, we would expect *always-inside* to yield true. We can distinguish different kinds of "forall" quantifications that result from different time intervals over which aggregation can be defined to range. In the case of *inside* the expected behavior is obtained if the aggregation ranges over the lifetime of the first argument, the moving point. This is not true for all spatial predicates. The chosen aggregation depends on the nature and use of each individual predicate. For example, two spatio-temporal objects are considered as being *equal* only if they are equal on both objects' lifetimes, that is, the objects must have the same lifespans and must be always equal during these.

In this sense, we have identified four different kinds of meaningful "forall" quantifications and associated the most suitable one to each spatio-temporal version of the eight basic spatial predicates for two regions; for details and the formal definitions see [Erwig and Schneider, 2002]. For the purpose of this chapter it is sufficient to know that the lifted spatial predicate has to be true for all times of the first, the second, the union, or the intersection of both objects' lifetimes as indicated in the following table:

	first	second	union	intersection
Disjoint, *Meet*, *Overlap*				×
Equal			×	
Covers, *Contains*		×		
CoveredBy, *Inside*	×			

The table defines the predicates for two evolving regions; for a moving point and a moving region we have just the three basic predicates *Disjoint*, *Meet*, and *Inside*, which are defined as above.

The defined spatio-temporal predicates are the basic building blocks of a language for specifying changes of spatio-temporal objects, called *developments* [Erwig and Schneider, 1999a]. In fact, with these basic predicates alone we cannot describe changes in the topological relationships of spatio-temporal objects. Therefore, we we need operations to combine them into more complex predicates. The most important operation is *composition*, written as "$\triangleright$". For example, the composition

$$\mathit{Disjoint} \triangleright \mathit{Meet}$$

defines a spatio-temporal predicate that yields true only for two objects that were disjoint for some time and after that met for some time. We can also compose spatial predicates with spatio-temporal ones. For example, the composition

$$\mathit{Disjoint} \triangleright \mathit{meet} \triangleright \mathit{Inside}$$

defines a spatio-temporal predicate that yields true for a moving point entering an evolving region. Between being outside and inside there is one instant of time when the point is on the border of the region.

When we consider in more detail how spatial situations can change over time, we observe that certain relationships can be observed only for a period of time and not for only a single time point (given that the participating objects do exist for a period of time) while other relationships can hold at instants as well as on time intervals. Predicates that can hold at time points and intervals are: *equal*, *meet*, *covers*, *coveredBy*; these are called *instant predicates*. Predicates that can only hold on intervals are: *disjoint*, *overlap*, *inside*, *contains*; these are called *period predicates*.

It is interesting to note that (in satisfiable developments) instant and period predicates always occur in alternating patterns, for example, there cannot be two spatio-temporal objects that satisfy *Inside* immediately followed by *Disjoint*. In contrast, *Inside* first followed by *Meet* (or *meet*) and then followed by *Disjoint* can be satisfied.

Sequential temporal composition is just one possibility to build new spatio-temporal predicates. Other operations are *alternative* and *reverse*. For example, consider a moving point on a border of a region. The situations that can arise when the point leaves the border are captured by the alternative *Disjoint*|*Inside*. An example for *reverse* will be shown in Section 4.

Finally, when we are interested only in an initial or a final part of a development, it is helpful to have a kind of “wildcard” spatio-temporal predicate that can be used to express “don’t care” parts of developments. Therefore, we have defined a spatio-temporal predicate *True* that yields always true for two spatio-temporal objects. By composing *True* with other spatio-temporal predicates we can achieve the effect of specifying “don’t care” parts of developments; an example is given in the next section.

4. Querying with Spatio-Temporal Operations

In this section we demonstrate by example queries how spatio-temporal data types and operations can be embedded into our *spatio-temporal query language* called *STQL*. The full extent of available operations, their signatures and semantics, as well as more advanced query constructs and facilities can be found in [Güting et al., 2000; Erwig and Schneider, 2002]. The intention here is to illustrate some new kinds of queries that can be posed against a spatio-temporal database.

4.1 Design Aspects and Application Scenarios

From a design point of view, our intention is not to devise a new spatio-temporal query language from scratch but to appropriately extend the widespread database query language standard SQL. We profit from the fact that the underlying data model rests on the ADT approach which necessitates only conservative extensions to SQL. These are essentially: (i) a set of spatial data types, operations, and predicates (taken for granted), (ii) a set of spatio-temporal operations (obtained for free by temporal lifting), (iii) temporal selection, (iv) projection operations to space and time, (v) spatio-temporal aggregation, (vi) a set of basic spatio-temporal predicates, and (vii) an extension mechanism for the construction of new, more complex spatio-temporal predicates.

The benefit of this approach is the preservation of well known SQL concepts, the high-level treatment of spatio-temporal objects, and the easy incorporation of spatio-temporal operations and predicates. Users can ask either standard SQL queries on standard data or use STQL features to inquire about situations involving spatial, temporal, or spatio-temporal data.

We will consider queries from three (simplified) application scenarios. The first scenario refers to a flight-weather information system. Flight and weather conditions play a central role for the feasibility of flights and the safety of passengers. Here we use the following relations:

```
flights(id:string, Route:Point)
weather(kind:string, Extent:Region)
```

`Point` and `Region` are the two types for moving points and evolving regions, respectively. The attribute `id` identifies a flight, and `Route` records the route of a flight over time. The attribute `kind` classifies different weather events like hurricanes, high pressure areas, or snowfall; `Extent` yields the evolving extent of each weather event.

The second scenario is related to forest fire control management which pursues the important goal of learning from past fires and their evolution. We assume a database containing relations with schemas

```
forest(forestname:string, Territory:Region)
forest_fire(firename:string, Extent:Region)
fire_fighter(fightername:string, Location:Point)
```

The relation `forest` records the location and the development of different forests (attribute `Territory`) growing and shrinking over time through clearing, cultivation, and destruction processes, for example. The relation `forest_fire` documents the evolution of different fires from their ignition up to their extinction (attribute `Extent`). The relation `fire_fighter` describes the motion of fire fighters being on duty from their start at the fire station up to their return (attribute `Location`).

The third scenario, finally, relates to a database about the migration of birds in order to explore their behavior patterns over the years.

```
birds (swarm:string, Movement:Point)
```

4.2 Temporal Selections

The first queries refer to the flight-weather information system. A *temporal selection* extracts the value of a moving object at a certain instant or the temporal development over a certain period. We can then ask queries like “Where was United Airlines flight 207 at time 8:00 am?”.

```
SELECT Route(8:00) FROM flights
 WHERE id = "UA207"
```

This query shows the functional character of a spatio-temporal object by determining the value of the object at a certain time through a simple function application. A more general version of this query asks where the plane was between 7:00 am and 9:00 am.

```
SELECT Route(7:00..9:00) FROM flights
 WHERE id = "UA207"
```

The “`..`” notation specifies a range of time values, that is, a time interval. If a spatio-temporal object is applied to a time interval (or a collection of disjoint time intervals separated by commas), this expression yields a spatio-temporal object restricted to that time interval (function restriction).

4.3 Projections to Space and Time

Projection operations on moving objects map either to their spatial or to their temporal aspect. Assume that we are interested in the geometric locations where the plane was between 7:00 am and 9:00 am. These can be obtained by:

```
SELECT trajectory(Route(7:00..9:00))
  FROM flights
 WHERE id = "UA207"
```

The `trajectory` operation computes the *spatial projection* of a spatio-temporal object onto the Euclidean plane. For a moving point it yields an object of the spatial type *line*. Note that isolated stationary points that can, in general, also occur are ignored. For an evolving region the trajectory operation returns an object of the spatial type *region* which results from projecting the union of the region values at all times onto the Euclidean plane.

The next query asks for the lifespan of a spatio-temporal object: “How long took the flight 207?”

```
SELECT duration(dom(Route)) FROM flights
 WHERE id = "UA207"
```

The `dom` operator collects the times when the flight UA207 is defined (*temporal projection*). In this way inverse temporal functions can be computed. The `duration` operation computes the length of an interval or of several intervals.

4.4 Aggregations

The following query inquires about the largest snow areas at all times.

```
SELECT Area(max(Extent)) FROM weather
 WHERE kind = "snow"
```

The query demonstrates an example of a *spatio-temporal aggregation* operation `max` which is an extension of the well known aggregation operator in SQL of the same name. It is here applied to a collection of evolving regions contained in a relation column and computes a new evolving region. Internally, this operator is based on a binary function max_{st} applied to two evolving regions R_1 and R_2 and yielding a new evolving region in the following way:

$$max_{st}(R_1, R_2) \ := \ \{(t, r) \mid t \in \textit{time} \ \wedge r = max_{geo}(R_1(t), R_2(t))\}$$

This definition uses a function max_{geo} which is applied to two regions r_1 and r_2 and which returns the larger of both regions:

$$max_{geo}(r_1, r_2) = \begin{cases} r_1 & \text{if } \textit{area}(r_1) > \textit{area}(r_2) \\ r_2 & \text{otherwise} \end{cases}$$

Altogether this means that for n evolving regions $R_1, \ldots, R_n$ we first compute the evolving region $R = max_{st}(R_1, \ldots, max_{st}(R_{n-1}, R_n) \ldots)$. Afterwards, we apply the lifted operator `Area` to R, which computes the area of R at all times as a temporal real number. Alternatively, we can answer the query by

```
SELECT max(Area(Extent)) FROM weather
 WHERE kind = "snow"
```

Here, first the `Area` operator is applied to each snow area and returns a temporal real. Then the `max` operator takes the collection of temporal reals and produces a new temporal real by selecting the largest of all *real* values occurring in the temporal reals at each time.

4.5 Temporally Lifted Operations

The concept of *temporal lifting* has been discussed in Section 3.2. It allows us to lift all spatial operations to the temporal dimension. The following examples are taken from the forest-fire control-management scenario.

The first query asks for the total size of the forest areas destroyed by the fire called “The Big Fire”.

```
SELECT sum(size) FROM
  (SELECT size AS area(traversed(
             Intersection(Territory, Extent)))
     FROM forest_fire, forest
    WHERE firename = "The Big Fire" AND
          ever(Intersects(Territory, Extent)))
```

The lifted predicate `Intersects` is part of the *spatio-temporal join* condition of the nested subquery. If the name of a forest fire is “The Big Fire” and if its extent overlaps with the territory of a forest at least at one time (`ever`), the intersection is computed by the lifted spatial operation `Intersection`. Finally, from the resulting evolving region the area of its spatial projection is determined.

The next query asks for the times and locations when and where the spread of fires was larger than 500 *km*^{2}.

```
SELECT Extent(dom(at(Area(Extent)>500,true)))
  FROM forest_fire
 WHERE not(isempty(dom(at(Area(Extent)>500,true))))
```

The `at` operation takes a lifted predicate (`Area(Extent) > 500`) and a boolean constant (`true`) as operands. The lifted predicate itself contains a lifted `Area` function which determines the areas of `Extent` over all times of its lifespan and thus produces a temporal real. The lifted predicate now computes a temporal boolean testing for each time of the lifespan of the temporal real whether the real value at that time is larger than 500 km^2 or not. The `at` operation then extracts those times and boolean values from the temporal boolean where the boolean value is true. The `isempty` predicate tests whether a set is empty.

Another query asks how long fire fighter Th. Miller was enclosed by the fire called “The Big Fire” and which distance he covered there.

```
SELECT time AS duration(dom(
             Intersection(Location, TheBigFire))),
       distance AS length(trajectory(
             Intersection(Location, TheBigFire)))
  FROM fire_fighter
 WHERE fightername = "Th. Miller"
```

We assume that the value `TheBigFire` has already been determined before, and that we know that Th. Miller was in this fire (otherwise time and distance will be returned as zero). This time, the `Intersection` operation is applied to a moving point (`Location`) and an evolving region (`TheBigFire`). It yields that part of the moving point lying inside the evolving region. The operation `length` determines the length of a line, which is here obtained as the result of the spatial projection of a moving point.

Finally, we give an example query using a lifted constant, that is, a non-moving spatio-temporal object. With respect to our flight-weather information system we ask when a plane was over the Eiffel Tower.

```
SELECT dom(Intersection(Route, ˆEiffelTower))
  FROM flights
```

We assume that `EiffelTower` describes a point containing the coordinates of the Eiffel Tower. In STQL the lifting operator is denoted by `ˆ`. It is here applied to a point as a non-temporal object and yields a stationary point over time, that is, a moving point does not move.

4.6 Querying Developments in STQL

We first give a motivation why spatio-temporal predicates are needed and then show some queries.

4.6.1 Motivation. In the Introduction we have emphasized that tuple-timestamped models are too inflexible for our purposes and that we prefer attribute-timestamped approaches. But even the latter approaches are restricted in the sense that they are incapable of modeling temporal developments of continuously evolving spatial relationships between moving objects. For example, consider the query "Determine the time when flight UA207 flew into a hurricane". We cannot express this query with languages like *TempSQL* [Gadia and Nair, 1993], which uses attribute timestamps, for two reasons. First, it is not possible in TempSQL to express and to query continuous developments, but only stepwise changes. Second, there is no interpolation mechanism telling when an event happened within a time interval. However, by employing our ADT approach, we can formulate the query as follows:

```
SELECT min(dom(Intersection(Route, Extent)))
  FROM flights, weather
 WHERE id = "UA207" AND
       kind = "hurricane"
```

The lifted `Intersection` operation is here applied to a moving point and an evolving region. It yields that part of `Route` lying inside `Extent`. The `min` operator here computes the minimum of all time values when the intersection is defined.

If we formulate this query a little bit more generally like "Determine the times when airplanes flew into hurricanes", a lot of plane-hurricane combinations might produce undesired null values because the flight and the hurricane just considered did not intersect at all. (Note that this is also possible in the more restricted previous query.) In this case, a spatio-temporal predicate could be used in the `WHERE` clause to avoid these null values in the result relation. Such a predicate could investigate in advance whether or not a flight and a hurricane came into contact. In the next section we will see how these predicates can be defined.

A second query showing the necessity of spatio-temporal predicates, in particular, for temporal developments, asks: "Determine the flights entering a hurricane". The problem here is that for each plane/hurricane combination we have to check the validity of different spatial predicates during a series of events and periods in a given temporal order. This means, we have to examine whether there has been a constellation when the plane and the hurricane were disjoint for a while, when afterwards they met at one point in time, and when finally the plane was inside the hurricane for a while. The development of entering a hurricane is only true if each of the three subqueries can be answered in the affirmative and if they have occurred one after the other. The series is like a specification that has to be matched at least once by each plane/hurricane combination. We can express this query by the following STQL statement:

```
SELECT id FROM flights, weather
 WHERE kind = "hurricane" AND
       not(Route(min(dom(Route))) inside
           Extent(min(dom(Route)))) AND
       Route(max(dom(Route))) inside
       Extent(max(dom(Route)))
```

Obviously, this query is very complicated. It works as follows: after the computation of the departure time of the flight (`min(dom(Route))`), the `Route` object is applied to this value and yields a point. The `Extent` object is applied to the same time value and yields a region. Using the spatial predicate `inside`, we check whether the point lies inside the region. If this is not true, we know that at the departure time of the flight the plane was outside of the hurricane. Similarly, we compute the arrival time of the flight (`max(dom(Route))`) and apply both `Route` and `Extent` to this time value. Again, we check whether the point lies inside the region, and if this is true, we know that `Route` must have entered the `Extent` object. This, in particular, implies that they met at the border of the hurricane. A limitation of this query is that we cannot determine whether plane and hurricane met only for one moment (straight entering) or whether the plane ran along the border for a while and then entered the hurricane (delayed entering). We will see how to express queries like these much more concisely in the following.

4.6.2 Querying. For integrating spatio-temporal predicates and developments into STQL, we extend it by (i) the set of eight basic spatio-temporal predicates and (ii) by a facility to assemble new complex predicates from more elementary ones.

We again consider the flight-weather information system. Let us first reconsider the example query of finding out all planes that ran into a hurricane. With a predicate combinator >> that has the semantics of temporal composition $\triangleright$ we can formulate the query as:

```
SELECT id FROM flights, weather
 WHERE kind = "hurricane" AND
       Route Disjoint>>meet>>Inside Extent
```

Since some compound predicates will be needed more frequently and since some of them have quite longish specifications, we introduce a macro definition facility as part of the data definition language to introduce new predicates. The syntax is given in Figure 1.1. As basic predicates (*p-basic*) we allow all the elementary spatio-temporal predicates introduced in Section 3.3.

We use the convention that | binds stronger than >> and that combinators >> and | bind stronger than predicate application. (This is the reason that we

p-def	$\rightarrow$	`DEFINE` *p-name* `AS` *p-expr*
p-expr	$\rightarrow$	*p-basic*
	$\mid$	*p-name*
	$\mid$	*p-expr* `>>` *p-expr*
	$\mid$	*p-expr* `|` *p-expr*
	$\mid$	`rev(`*p-expr*`)`

Figure 1.1. Predicate definition macro language.

were able omit the brackets around the spatio-temporal predicate in the above example query.)

Now we can define a predicate `Enters` as follows:

```
DEFINE Enters AS Disjoint>>meet>>Inside
```

Hence, we can formulate the query asking for planes entering a hurricane also as:

```
SELECT * FROM flights, weather
 WHERE kind = "hurricane" AND
       Route Enters Extent
```

As further examples, consider the definition of the predicates `Leaves`, `Crosses`, and `Bypass`:

```
DEFINE Leaves AS rev(Enters)
DEFINE Crosses AS Enters>>Leaves
DEFINE Bypasses AS Disjoint>>Meet>>Disjoint
```

Note that the predicate `Crosses` is equal to the definition

```
DEFINE Crosses
    AS Disjoint>>meet>>Inside>>meet>>Disjoint
```

because `rev(Enters)` = `Inside>>meet>>Disjoint` and `Inside>> Inside` = `Inside`. General laws expressing relationships like these are given in [Erwig and Schneider, 2002].

The following example illustrates the construct of alternative. The query is to find all planes that either crossed or bypassed a snowstorm.

```
SELECT id FROM flights, weather
WHERE kind = "snowstorm" AND
      Route Crosses|Bypasses Extent
```

We can use development predicates also within `GROUP BY` clauses (*spatio-temporal grouping*). For example, we might be interested in the number of planes that were, respectively, were not entering snowstorms or fog areas:

```
    SELECT COUNT(*) FROM flights, weather
     WHERE kind = "snowstorm" OR kind = "fog"
  GROUP BY Route Enters Extent
```

To demonstrate the use of developments on two evolving regions we switch to the forest0-fire control-management scenario. We could be interested, for example, in all forests that were completely destroyed by a particular fire. The fact that a forest is destroyed means that it is, at least from some time on, completely inside of (or equal to) the fire region, that is, after the fire is over, the forest does not exist anymore. But before that many different relationships between the fire and the forest are possible, for example, the fire ignition can happen within the forest, at its border or outside. Since we do not want to care about all these possibilities, we can use the predicate *True* as a wildcard preceding the final condition, which we denote in STQL by _. This leads to the following query:

```
SELECT name FROM forest, fire
 WHERE Territory _>>Inside|Equal Extent
```

This means that for a certain period of time we do not care at all about the relationship between the forest and the fire, which is expressed by _ that constantly yields *true*; we only require the existence of a time point after which `Inside` or `Equal` holds.

Finally, as an example for querying spatio-temporal developments of two moving points, consider the relation recording the migration of birds.

We might be interested in swarms that fly together, then take different routes for some time and finally meet again. This can be expressed as an STQL query:

```
DEFINE Remeets
    AS _>>Meet>>Disjoint>>Meet>>_

SELECT A.swarm, B.swarm
  FROM birds (A), birds (B)
 WHERE A.Movement Remeets B.Movement
```

5. Visual Querying

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, in particular, 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 can then be automatically translated into queries using spatio-temporal predicates [Erwig and Schneider, 2000; Erwig and Schneider, 1999b]. A sketch for the spatio-temporal predicate *Inside* ▷ *meet* ▷ *Disjoint* is shown in Figure 1.2.

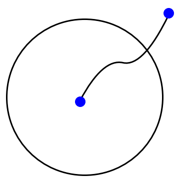

Figure 1.2. A visual predicate specification.

There are at least two ways in which these visual predicate sketches can be utilized for spatio-temporal query languages: (1) Sketches can be embedded directly into queries to replace spatio-temporal predicates. This approach leads to a heterogeneous visual languages as investigated in [Erwig and Meyer, 1995]. On the other hand, we can extend the visual notation by allowing textual attachments to graphical objects. The text fragments are meant to identify objects in the database that have an attribute that correspond to the sketched trace. With these additional specifications we are able to translate sketches into complete queries.

For example, consider the visual query in Figure 1.3.

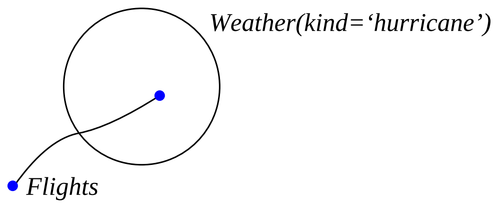

Figure 1.3. A visual query.

The names associated with the graphical objects are interpreted as the relations whose spatio-temporal objects are depicted (here, `Route` and `Extend`), and the condition in brackets is treated as a condition that is just added to the `WHERE` clause. Together with the translation of the depicted trace, the visual query can be translated into the query given at the beginning of Section 4.6.2.

6. Conclusions

Based on an ADT approach to the integration of spatio-temporal data types into data models we have shown how to extend SQL to a spatio-temporal query language called STQL. We have shown how to obtain query operators by temporal lifting and how to express temporal selections and aggregations. A distinctive feature of the ADT approach is that ADT operations can be integrated smoothly into SQL so that existing query mechanisms like grouping can be used together with the new operations. Observing that querying developments of spatial objects is of particular interest, we have demonstrated how to define basic and compound spatio-temporal predicates as specifications for developments textually and visually.

References

Ariav, G. (1986). An Overview of TQuel. *ACM Transactions on Database Systems*, 11(4):499–527.

Böhlen, M. H., Jensen, C. S., and Skjellaug, B. (1998). Spatio-Temporal Database Support for Legacy Applications. In *ACM Symp. on Applied Computing*, pages 226–234.

Cheng, T. S. and Gadia, S. K. (1994). A Pattern Matching Language for Spatio-Temporal Databases. In *ACM Conf. on Information and Knowledge Management*, pages 288–295.

Clementini, E., Felice, P., and Oosterom, P. (1993). A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In *3rd Int. Symp. on Advances in Spatial Databases*, LNCS 692, pages 277–295.

Clifford, J. and Croker, A. (1993). *The Historical Relational Data Model (HRDM) Revisited*, pages 6–27. In [Tansel et al., 1993].

Clifford, J., Croker, A., and Tuzhilin, A. (1993). *On the Completeness of Query Languages for Grouped and Ungrouped Historical Data Models*, pages 496–533. In [Tansel et al., 1993].

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

Egenhofer, M. J. and Al-Taha, K. K. (1992). Reasoning about Gradual Changes of Topological Relationships. In *Int. Conf. GIS – From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, LNCS 639, pages 196–219.

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

Egenhofer, M. J. and Herring, J. (1990). A Mathematical Framework for the Definition of Topological Relationships. In *4th Int. Symp. on Spatial Data Handling*, pages 803–813.

Erwig, M. and Güting, R. H. (1994). Explicit Graphs in a Functional Model for Spatial Databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):787–804.

Erwig, M., Güting, R. H., Schneider, M., and Vazirgiannis, M. (1998a). Abstract and Discrete Modeling of Spatio-Temporal Data Types. In *6th ACM Symp. on Geographic Information Systems*, pages 131–136.

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

Erwig, M. and Meyer, B. (1995). Heterogeneous Visual Languages – Integrating Visual and Textual Programming. In *11th IEEE Symp. on Visual Languages*, pages 318–325.

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

Erwig, M. and Schneider, M. (1999b). Visual Specifications of Spatio-Temporal Developments. In *15th IEEE Symp. on Visual Languages*, pages 187–188.

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

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

Erwig, M., Schneider, M., and Güting, R. H. (1998b). Temporal Objects for Spatio-Temporal Data Models and a Comparison of Their Representations. In *Int. Workshop on Advances in Database Technologies*, LNCS 1552, pages 454–465.

Gadia, S. K. and Nair, S. S. (1993). *Temporal Databases: A Prelude to Parametric Data*, pages 28–66. In [Tansel et al., 1993].

Grumbach, S., Rigaux, P., and Segoufin, L. (1998). Spatio-Temporal Data Handling with Constraints. In *6th ACM Int. Symp. on Advances in Geographic Information Systems*, pages 106–111.

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

Güting, R. H. and Schneider, M. (1995). Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal*, 4(2):100–143.

Motakis, I. and Zaniolo, C. (1995). Composite Temporal Events in Active Databases: A Formal Semantics. In Clifford, J. and Tuzhilin, A., editors, *Recent Advances in Temporal Databases*, pages 332–351. Springer Verlag.

Segev, A. and Shoshani, A. (1993). Logical Modeling of Temporal Data. *In [Tansel et al., 1993]*, pages 248–270.

Snodgrass, R. (1993). *A Temporally Oriented Data Model*, pages 141–182. In [Tansel et al., 1993].

Stonebraker, M. (1986). Inclusion of New Types in Relational Database Systems. In *Int. Conf. on Data Engineering*, pages 262–269.

Stonebraker, M., Rubenstein, B., and Guttman, A. (1983). Application of Abstract Data Types and Abstract Indices to CAD Data Bases. In *ACM/IEEE Conf. on Engineering Design Applications*, pages 107–113.

Tansel, A. U., Clifford, J., Gadia, S., Jajodia, S., Segev, A., and Snodgrass, R., editors (1993). *Temporal Databases: Theory, Design, and Implementation*. The Benjamin/Cummings Publishing Company.

Worboys, M. F. (1994). A Unified Model for Spatial and Temporal Information. *The Computer Journal*, 37(1):25–34.

Yeh, T. S. and Cambray, B. (1993). Time as a Geometric Dimension for Modeling the Evolution of Entities: A 3D Approach. In *Int. Conf. on Integrating GIS and Environmental Modeling*.

Yeh, T. S. and Cambray, B. (1995). Modeling Highly Variable Spatio-Temporal Data. In *6th AustraliAsian Database Conf.*, pages 221–230.