Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Visual Languages and Computing
14 (2003) 181–211
Journal of Visual Languages & Computing
www.elsevier.com/locate/jvlc

A visual language for the evolution of spatial relationships and its translation into a spatio-temporal calculus

Martin Erwig[a,*], Markus Schneider[b]

[a] *Department of Computer Science, Oregon State University, Corvallis, OR 97331, USA*
[b] *University of Florida, Computer & Information Science & Engineering, Gainesville, FL 32611-6120, USA*

Received 5 April 2001; received in revised form 13 June 2002; accepted 17 June 2002

Abstract

Queries about objects that change their spatial attributes over time become particularly interesting when they ask for changes in the spatial relationships between different objects. We propose a visual notation that is able to describe scenarios of changing object relationships. The visual language is based on the idea to analyze two-dimensional traces of moving objects to infer a temporal development of their mutual spatial relationships. We motivate the language design by successively simplifying object traces to their intrinsic properties. The notation can be effectively translated into a calculus of spatio-temporal predicates that formally characterizes the evolution of spatial relationships. We also outline a user interface that supports specifications by menus and a drawing editor. The visual notation can be used directly as a visual query interface to spatio-temporal databases, or it can provide predicate specifications that can be integrated into textual query languages leading to heterogeneous languages.

Keywords: Visual predicate specification; Visual database interface; Spatio-temporal queries; Object traces; Translation

*Corresponding author.
E-mail addresses: erwig@cs.orst.edu (M. Erwig), mschneid@cise.ufl.edu (M. Schneider).

1045-926X/03/$ - see front matter © 2003 Elsevier Science Ltd. All rights reserved.
doi:10.1016/S1045-926X(02)00057-5

1. Introduction

Stimulated by the deep relationships between temporal and spatial data there has recently been an increasing interest in integrating database support for both kinds of data and in designing *spatio-temporal databases*. Spatio-temporal databases deal with spatial objects that change over time (for example, they move, they grow, or they alter their shape): cars, planes, people, animals, ..., storms, lakes, forests, etc. Hence, database systems, in particular, spatial and temporal database systems, and geographical information systems (GIS) need to be appropriately extended to handle this kind of information. Of particular interest is, of course, the development of simple but powerful query languages that allow one to ask for changes in spatial relationships, for instance: 'Has a tornado ever crossed Iowa?' or 'Which planes were able to avoid a certain blizzard?'. A formal foundation for these kinds of queries is given by the concept of *spatio-temporal predicates* [1]. Whereas it is possible to identify a relatively small set of spatial predicates [2], it is almost impossible to do so in the spatio-temporal case, simply because there are too many of them. Thus, there is a very strong need for a simple way of specifying spatio-temporal situations, and a visual notation can be extremely helpful here.

This paper continues our previous work [3,4] aiming at proposing a visual language for spatio-temporal predicates. The main idea is to graphically represent the temporally changing evolution of a spatio-temporal object (like a car or a storm) in a two-dimensional way by its trace. The topological behavior of such a trace with respect to another object is interpreted and translated into a sequence of predicates, called *development*, that can then be used, for example, to query spatio-temporal databases. In this paper, we especially provide a precise description of the translation process of a visual specification into a predicate sequence. Moreover, we will also deal with partial specifications, which we have not considered in our previous work.

The described visual notation can be employed in several ways. One application is, as already mentioned, to realize a visual query interface to spatio-temporal databases. But we can also use pictures of this language as specifications for (complex) spatio-temporal predicates, which can then be used in arbitrary query languages. One interesting possibility is to use a well-accepted textual query language like SQL, to extend it by spatio-temporal objects and predicates [5], and to use pictures to represent predicates in WHERE clauses. This leads then to a heterogeneous visual language [6].

The paper is structured as follows: Section 2 comments on related work. In Section 3, from a user interface and application point of view, we sketch a visual query interface, called *Query-By-Trace*, that is capable of specifying, querying and evaluating developments (that is, changing object relationships) in spatio-temporal databases. Section 4 describes how spatio-temporal data can be modeled. In particular, we explain the notions of spatio-temporal objects, predicates and developments. In Section 5, from a design point of view, we then explain and motivate the concept of our visual notation for developments. The interpretation of the visual notation and the translation into predicate sequences is then defined in Section 6. Finally, conclusions are given in Section 7.

2. Related work

The affinity of spatial and temporal phenomena has been recognized for a long time in the literature. Both phenomena deal with 'spaces' or 'dimensions' of some kind and are thus closely related. Recently, research efforts have led both in spatial and in temporal data modeling to an increased interest in integrating both directions into a common research branch called *spatio-temporal data modeling* and in constructing *spatio-temporal databases*. Their underlying basic entities are called *spatio-temporal objects* and are ubiquitous in everyday life. Consider the flight of an airplane, the migration of whales, the raging of a storm, or the spreading of a fire region. Characteristic features of all these objects are that they are *spatial entities changing over time* and that these changes are *continuous*. Changes refer, for example, to motion, shrinking, growing, shape transformation, splitting, merging, disappearing or reappearing of spatio-temporal objects. In particular, the capability of incorporating continuous change of spatial objects over time belongs to the most challenging requirements of spatio-temporal data models.

In the meanwhile, some data models for spatio-temporal databases have already been proposed. In [7] a spatial data model has been generalized to become spatio-temporal: spatio-temporal objects are defined as the 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 [8] which are described in [9,10]. The main drawback of all these approaches is that ultimately they are incapable of modeling continuous changes of spatial objects over time.

Our approach to dealing with spatio-temporal data takes a more integrated view of space and time and includes the treatment of continuous spatial changes. It introduces the concept of *spatio-temporal data types* [11,12]. These data types are designed as *abstract data types* whose values can be integrated as complex entities into databases [13,14] and whose definition and integration into databases is independent of a particular DBMS data model.

The definition of a temporal object [15] in general is motivated by 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 are regarded as special instances of temporal objects where α is a spatial data type like *point* or *region*. A point (representing an airplane, for instance) that changes its location in the Euclidean plane over time is called a *moving point*. Similarly, a temporally changing region (representing a fire area, for instance) is a region that can move and/or grow/shrink and whose components can split or merge. We call such an object an *evolving region*.

A straightforward and very instructive view of spatio-temporal objects is to visualize their temporal evolution as purely geometric, *three-dimensional* objects, that is, the time axis is regarded as an additional third geometric dimension. An evolving region is then represented as a *volume* in 3D space, and a moving point is then

visualized as a *3D curve*. Any intersection with the *xy*-plane yields a spatial object, that is, a region or a point. This observation is the starting point for the design of our visual language: by successively abstracting from irrelevant information we eventually arrive at a two-dimensional query language.

Similar to our approach, in [16,17] based on the work in [18] the so-called *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 between two consecutive elements of the sequence. Whereas this approach mainly focuses on representational issues and advocates the three-dimensional object view of spatio-temporal objects, we are particularly interested in an algebraic model of general spatio-temporal data types including a comprehensive collection of spatio-temporal operations [12,19]. Nevertheless, behavioral time sequences could be used as representations for our temporal objects.

Temporal changes of spatial objects induce modifications of their mutual topological relationships over time. For example, at one time two spatio-temporal objects might be disjoint whereas some time later they might intersect. These modifications usually proceed continuously over time but can, of course, also have a discrete property. We already have devised and formally defined a concept for such *spatio-temporal relationships* which are described by the so-called *spatio-temporal predicates* [1]. We call a sequence of spatial and spatio-temporal predicates a *development*.

Since we are dealing with predicates, it is not surprising that logic-based approaches are related to our work. Allen [20] defines a predicate $Holds(p, i)$ which asserts that a property p is true during a time interval i. Galton [21] has extended Allen's approach to the treatment of temporally changing two-dimensional topological relationships. Topological predicates are taken from the RCC model [22] which comes to similar results as Egenhofer's 9-intersection model, which is briefly discussed below. In contrast to these approaches, we have pursued a hybrid approach taking into account elements from temporal logic and elements from point-set theory and point-set topology. The main reason for not taking a purely logic approach is the intended integration of spatio-temporal objects and predicates into spatio-temporal databases and query languages. These require concrete representations for spatio-temporal objects and besides predicates the possibility of constructing new objects through spatio-temporal operations. Hence, efficiency, in particular for the evaluation of spatio-temporal queries, is indispensable.

Our work on spatio-temporal predicates is based on Egenhofer's 9-intersection model [2] for topological predicates between spatial objects in two-dimensional space. The goal of this model is to provide a canonical collection of topological relationships for each combination of spatial types. The model rests on the nine possible intersections of boundary, interior and exterior of a spatial object with the corresponding parts of another object. Each intersection is then tested with regard to the topologically invariant criteria of emptiness and non-emptiness. From the total number of possible topological constellations only a certain subset makes sense depending on the combination of spatial objects just considered. For two regions,

eight meaningful constellations have been identified which lead to the eight predicates called *equal*, *disjoint*, *coveredBy*, *covers*, *intersect*, *meet*, *inside* and *contains*. For a point and a region we obtain the three predicates *disjoint*, *meet* and *inside*. For two points we get the two predicates *disjoint* and *meet* (which corresponds to equality). For each group all predicates are mutually exclusive. They are also complete in the sense that they cover all possible topological constellations under the assumptions of the 9-intersection model.

There exist several approaches to visual query languages for spatial databases, for example, [23–29]. Common to all these approaches is that they allow to query only *static* spatial situations, that is, they can express queries like 'Retrieve all airports in Ohio'. A characteristic of the involved objects 'airport' and 'Ohio' is that these objects rarely change their location and/or extent. There are also a few approaches to querying image sequences [30–32]. However, the goal of these proposals is mainly to facilitate queries on video databases and not the querying of spatial (or spatio-temporal) databases. Since video data is largely unstructured (just a sequence of images), all these approaches have to be concerned with additional symbolic representations for the stored images to enable queries. Our visual notation is translated into sequences of predicates that can be directly used for querying the database representations of the spatio-temporal objects. A short, preliminary proposal of the visualization idea that is developed in this paper has been presented in [3]. The main concepts of a user interface for visually specifying spatio-temporal developments have been described in [4].

3. Query-by-trace: a GUI for specifying spatio-temporal developments

Our visual notation is demonstrated by a few application scenarios illustrating how this notation can be employed for querying developments in spatio-temporal databases. We give a rough outline of the interaction a user may perform when visually specifying queries. Our goal is to interactively and graphically produce a sketch from which a spatio-temporal predicate can be derived.

The user interface *Query-By-Trace* (*QBT*) allows a comfortable specification of developments. It incorporates an editor component to draw specifications. The horizontal dimension is the x-axis; the vertical dimension describes time. The top of the editor provides two menus, one for moving points and one for evolving regions. Assuming a relational setting, both menus show the available attributes related to spatio-temporal objects in the database together with the corresponding relation names in brackets. In our example we use an environmental database containing weather and flight information.

The first application scenario deals with a user's query asking for all flights crossing hurricanes, that is, the user is interested in the topological relationships between an evolving region and a moving point over time. The user selects from the menu *Evolving Regions* the attribute *extent* of the relation *hurricanes* (see Fig. 1) and clicks at a desired position on the canvas of the editor. The result of this action is a circle labeled with the name of the selected relation (see Fig. 2).

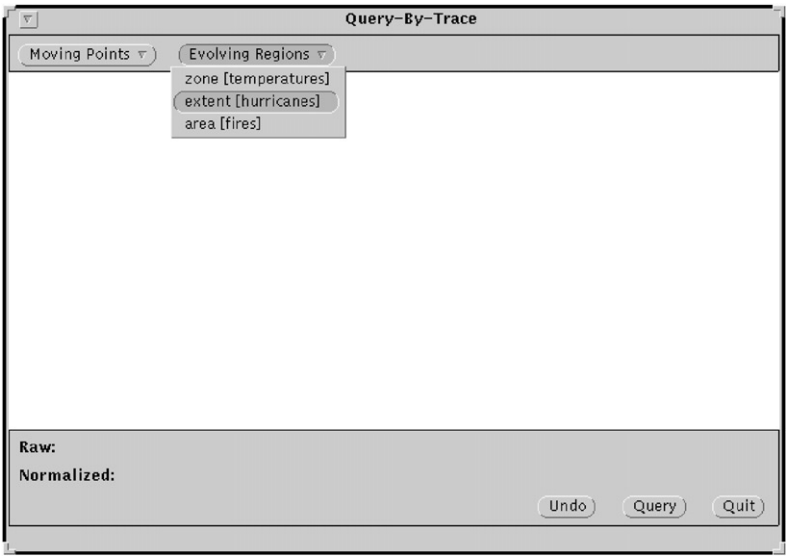

Fig. 1. Selecting the first object.

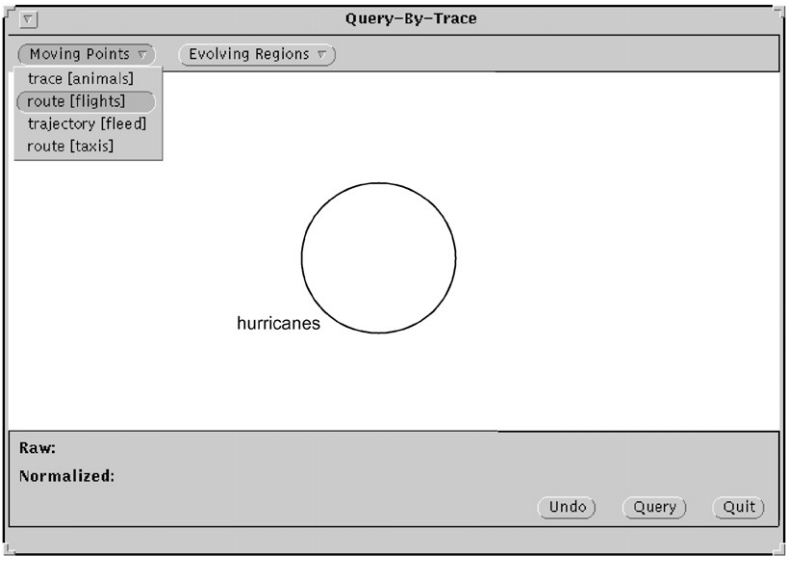

Fig. 2. Selecting the second object.

The user now selects from the menu *Moving Points* the attribute *route* of the relation *flights*. This indicates that the user is interested in specifying the development between an evolving region and a moving point. The user next draws

a crossing situation, which requires the following interactions: a click at a desired position outside the circle produces the starting point. The system determines the initial relationship of this point with respect to the evolving region and displays the spatial predicate *disjoint* in the two message lines at the bottom of the editor. Now the user drags the mouse from bottom to top from the starting point towards the circle. Note that during the specification process it is not possible to drag the mouse cursor below the current position since a spatial object cannot move backward in time. As soon as the mouse cursor leaves the starting point, the name of the spatio-temporal predicate *Disjoint* is added to the message lines. The fact that a spatial predicate is constant for a certain period is registered in the message lines by a spatio-temporal predicate indicated by an initial capital letter (for example, *Disjoint* or *Meet*).

We distinguish between the *raw mode* and the *normalized mode* of a development specification. The raw mode corresponds to the original definition of a development as an alternating sequence of spatial and spatio-temporal predicates (see Section 4.3). The normalized mode introduces simplifications to make the specification more readable for the user. One of these simplifications is that a spatial predicate (like *disjoint*) followed or preceded by its corresponding spatio-temporal predicate (like *Disjoint*) can be abbreviated to the spatio-temporal predicate. Hence, in the second message line we only see the *Disjoint* predicate so far (see Fig. 3).

While moving the mouse, the system draws the trace of the point and steadily watches for possible changes in the topological relationship. Each change is recorded in the message lines. The user now continues to move the cursor toward the circle and then traverses it. So far the user has specified an *Enter* situation, that is, the

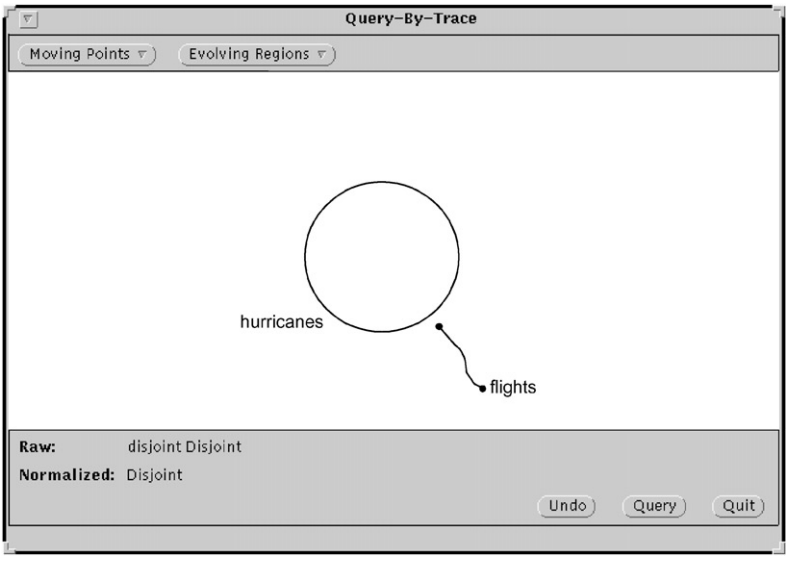

Fig. 3. Raw and normalized modes.

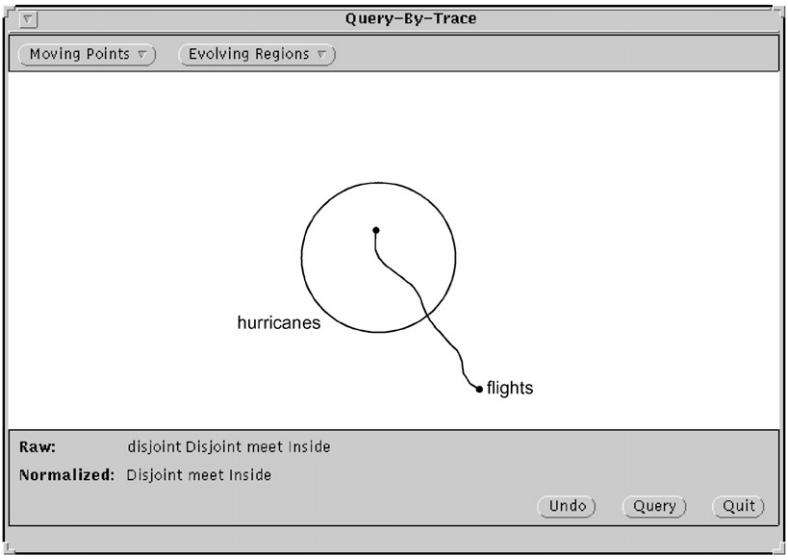

Fig. 4. QBT-visualization of an *Enter* situation.

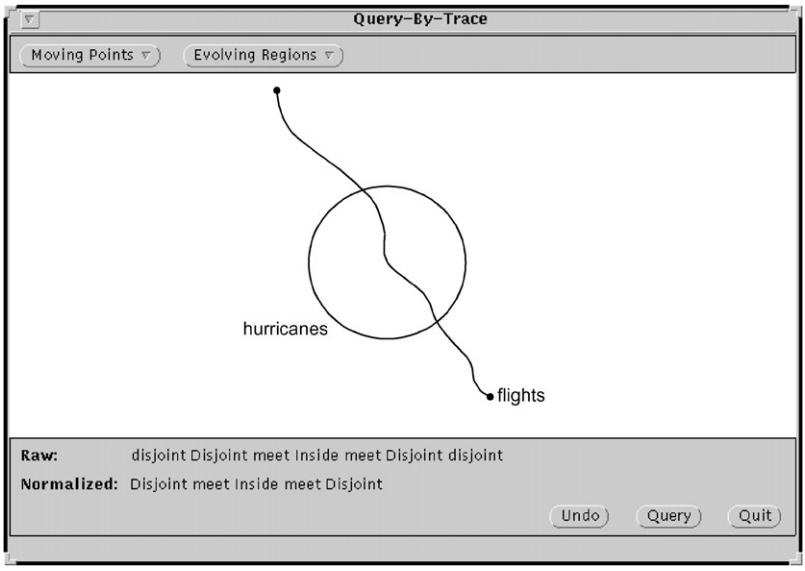

Fig. 5. Final QBT specification of the *Cross* predicate.

moving point at some time has met the circle and has been inside the circle since then (see Fig. 4). Afterwards the user drags the mouse to an end point outside the circle and releases the mouse button. The final picture is shown in Fig. 5 and specifies a

Cross situation. At the end of a dragging transaction, both the visual specification and the spatio-temporal predicate sequence are immediately available.

The scenario just described gives an example of the visual specification of the topological behavior between two spatio-temporal objects that is *completely* defined and known from its beginning up to its end; on the screen this is visible by the user's action drawing a continuous curve without releasing the mouse button in between. But we can also deal with the situation where the topological relationships between two spatio-temporal objects are unknown or irrelevant for certain periods of time. In that case a moving point could be, for example, inside an evolving region. Then the specification is interrupted at time t_1 for a certain period and afterwards continued at time t_2 ($> t_1$) within, on the border of, or outside the evolving region. What happens between t_1 and t_2 is then unknown or irrelevant. In both cases the user has to interactively release the mouse button and to press it again at another location.

If the user releases the mouse button, three different situations can arise: first, the user has completed the predicate specification. Second, the user has briefly interrupted the predicate specification and continues it at the current end point of the curve. Then the drawn end point disappears, and the drawing of the curve is resumed. Third, the user intends to specify a period of lacking knowledge or arbitrary topological behavior. In the predicate sequence this is indicated by the predicate *True* which has the function of a wildcard. Interactively, the user clicks at a mouse position which lies above the imaginary horizontal line and which does not necessarily have to describe a different topological relationship like the one at the last end point. A mouse click above the horizontal line is not subject to any restrictions.

The second application scenario deals with a user's query asking for a particular topological configuration between fires and hurricanes, that is, between two evolving regions. Moreover, this scenario gives an example of an only partially known topological behavior between two spatio-temporal objects. The user first selects the attribute *area* of the relation *fires* from the menu *Evolving Regions* and clicks at a desired position on the canvas. The result is a labeled circle appearing on the screen. Now the user selects the attribute *extent* from the relation *hurricanes* and positions a second smaller circle outside and below the larger circle on the screen.

The second circle can be moved and is smaller than the first one. The trace consists of two disjoint curves spanning a corridor which the moved circle traverses while being dragged from a start position to its end position (see Fig. 6). The two predicates *meet* and *coveredBy* (describing the situations when the circles touch externally, respectively, internally) need a particular interaction. They are called *instant predicates* since only they can be valid at an instant. They can, of course, also be valid for some period (*Meet*, *CoveredBy*). To distinguish these two cases interactively, the drawing of *Meet* and *CoveredBy* is supported by holding down the shift-key during dragging. The movement of the mouse is then restricted to go along the border of the constant object until the shift-key is released again.

In our scenario the user next draws an entering situation by dragging the mouse from bottom to top into the circle and then releasing the mouse button (see Fig. 6). A small circle marking the current end position of the trace appears on the screen.

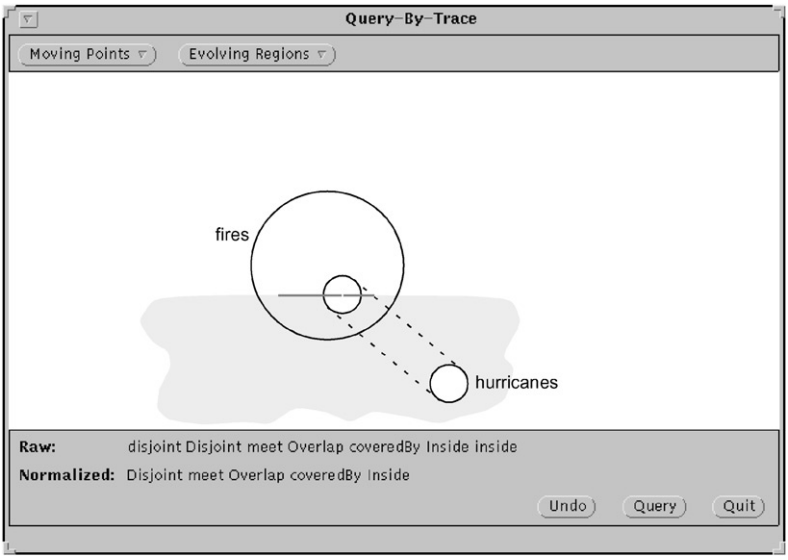

Fig. 6. Final QBT specification of the *Enter* predicate.

Again we can distinguish three different situations. First, the user has finished his specification.

Second, the user briefly interrupts the specification and then continues it at the current end position by clicking into the last drawn small circle. As a result, the circle disappears, and the drawing of the trace is resumed.

Third, the user intends to specify a period of lacking knowledge or arbitrary topological behavior which is indicated by the predicate *True* in the predicate sequence. Interactively, the user clicks at a mouse position which lies above an imaginary horizontal line through the center of the last drawn small circle and which does not necessarily have to describe a different topological relationship like the one at the last small circle. A positioning above the horizontal line is not subject to any restrictions. Note that the locations of the centers of circles are decisive and not the locations of the circles themselves. Hence, it may happen that a center of a circle is above a center of another circle but that the circles intersect each other.

In our scenario, as an illustration of the third case, the user next positions the mouse cursor above the last drawn small circle and outside the large circle. A small circle is drawn, and the user drags the circle over the boundary of the large circle and releases the mouse button when the mouse cursor is outside the large circle. This partial specification describes a *Graze* situation so that in the end we obtain the predicate *Enter True Graze* whose composing elementary predicate sequences are shown in raw and normalized mode in Fig. 7.

As a final application scenario, assume that we have a relation *birds* 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, that is, we are interested in the

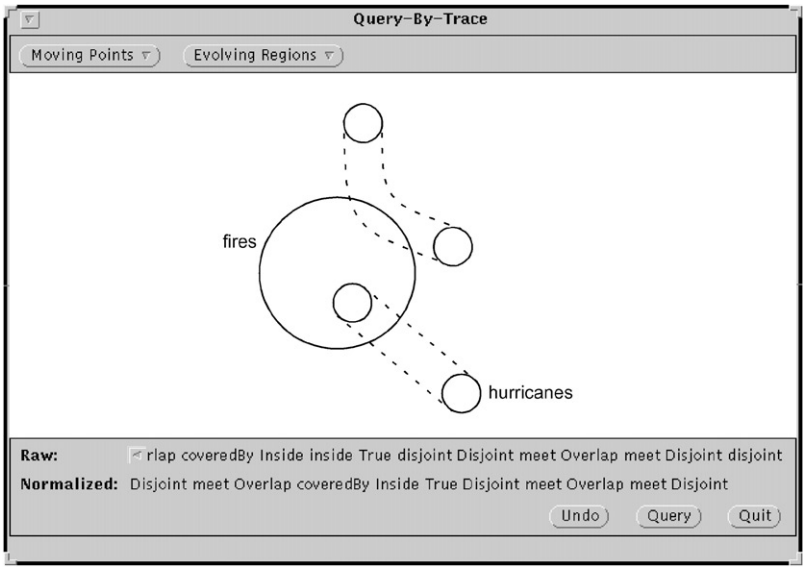

Fig. 7. QBT specification of the *Enter True Graze* predicate.

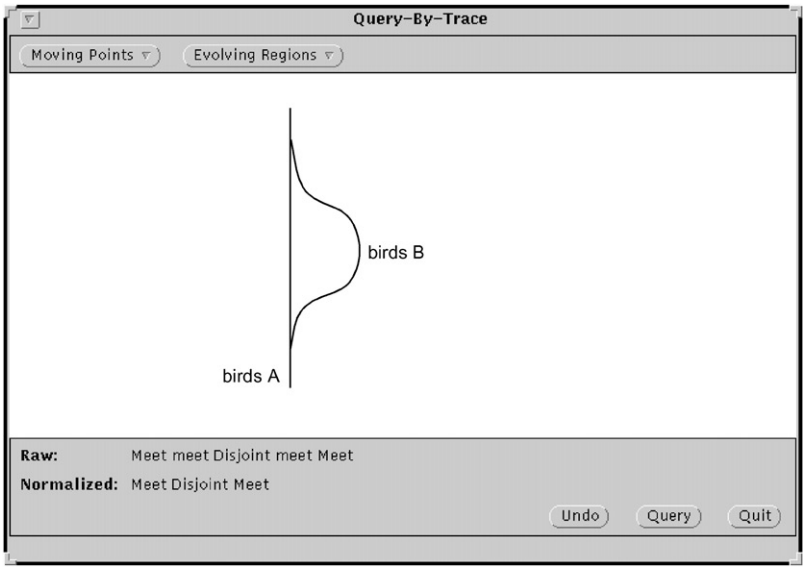

Fig. 8. Example of a QBT specification of two moving points.

topological relationships between two moving points over time. This can be expressed by the visual specification shown in Fig. 8. We see that if instead of a moving region a moving point is selected at the beginning, automatically a vertical

line is drawn which is static. In this situation the user is only allowed to select another moving point so that a development between two moving points is specified. This is because the specification of a development between a moving point and an evolving region should be performed on first selecting a static region and then a moving point.

We believe that this user interface is intuitive and easy to use because the user acts (via the mouse) as a moving object that behaves exactly in the way as the drawn spatio-temporal predicate demands it. In other words, the user action precisely conforms to, or satisfies, the specification that is drawn. Of course, to obtain reliable information about usability we have to perform a controlled user study that is beyond the scope of this paper.

4. Spatio-temporal objects, predicates and developments

In this section we will briefly review some of the formal foundations. We sketch our definition of spatio-temporal objects in Section 4.1, present our concept of spatio-temporal predicates in Section 4.2, and explain our specification mechanism for spatio-temporal developments in Section 4.3.

4.1. Spatio-temporal objects

One of our design goals is to define a spatio-temporal data model that is independent of a specific DBMS data model. This is achieved by encapsulating spatio-temporal data types into abstract data types which comprise a comprehensive collection of operations and predicates. Assuming a relational setting, for instance, we can then embed spatio-temporal data types in the same way like types for integers, reals, booleans or strings as attribute types in a relation, that is, the relation has only a container function to store attribute data in tuples.

The design of our 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, $time=\mathbb{R}$. The temporal version of a value of type α that changes over time can be modeled as a *temporal function* of type

$$\tau(\alpha) = time \to \alpha$$

We have used temporal functions as the basis of an algebraic data model for *spatio-temporal data types* [11] 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 $\tau(point)$ and is called a *moving point*. Similarly, an element of type $\tau(region)$ is a region that can move and/or grow/shrink. It is called an *evolving region*. 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 [12]. In any case, however, it is principally possible to integrate moving lines in much the same way as moving points if needed. In

addition, we also have changing numbers and booleans, which are essential when defining operations on temporal objects. 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:

$$Distance: \tau(point) \times \tau(region) \to \tau(real)$$

The example demonstrates the concept of *temporal lifting* [19] avoiding an inflation of operation names and definitions: we can, in principle, take almost any non-temporal operation (like $Distance: point \times region \to real$) and 'temporally lift' it so that it works on temporal objects returning also a temporal object as a result. More precisely, for each function $f: \alpha_1 \times \cdots \times \alpha_n \to \beta$ its corresponding lifted version

$$\uparrow f: \tau(\alpha_1) \times \cdots \times \tau(\alpha_n) \to \tau(\beta)$$

is defined by

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

Hence, we can derive temporal operations rather automatically. For example, we obtain $Distance = \uparrow distance$.

4.2. Spatio-temporal predicates

Temporal lifting is, of course, also applicable to spatial predicates. Consider the spatial predicate

$$inside: point \times region \to bool$$

The lifted version of this predicate has the type

$$\uparrow inside: \tau(point) \times \tau(region) \to \tau(bool)$$

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. We see that the lifted version is not a predicate since it yields a temporal boolean and not a (flat) boolean what we would expect from a predicate.

Our understanding of spatio-temporal predicates is the following: 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 basic 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 $\tau(\alpha) \times \tau(\beta) \to bool$ for $\alpha, \beta \in \{point, region\}$.

If we consider 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 and is always inside the region, we would expect *always-inside* to yield true.

In fact, 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. Actually, it depends on the nature and use of each individual predicate. For example, two spatio-temporal objects are considered as being *always-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 order to be able to concisely define spatio-temporal predicates, we use the syntax $Q_{op}.p$ to denote spatio-temporal predicates. $Q \in \{\forall, \exists\}$ is a quantifier, $op \in \{\cap, \cup, \pi_1, \pi_2\}$ is a function mapping two sets into a new set (π_i simply takes the ith argument set), and p is a spatial predicate. An expression $Q_{op}.p$ then denotes the spatio-temporal predicate:

$$\lambda(S_1, S_2).Qt \in op(dom(S_1), dom(S_2)).p(S_1(t), S_2(t)).$$

In general, $\lambda(x_1, x_2, \ldots).e$ denotes a function that takes arguments $x_1, x_2, \ldots$ (here two spatio-temporal objects S_1 and S_2) and returns a value determined by the expression e (here a boolean predicate quantified over time). This means that, for example, $\forall_{\pi_1}.inside$ denotes the spatio-temporal predicate

$$\lambda(S_1, S_2).\forall t \in dom(S_1).inside(S_1(t), S_2(t)).$$

This expression describes a function that takes S_1 and S_2 as arguments and checks if for all times t of the lifespan of S_1, the value of S_1 at t, which is a spatial object, lies inside S_2 at t, which also describes a spatial object. If this check yields true for all times t under consideration, the whole function yields true and otherwise false.

With this notation we can give the definitions for the spatio-temporal versions of the eight basic spatial predicates (for two regions):

$$\begin{aligned}
Disjoint &:= \forall_{\cap}.disjoint\\
Meet &:= \forall_{\cup}.meet\\
Overlap &:= \forall_{\cup}.overlap\\
Equal &:= \forall_{\cup}.equal\\
Covers &:= \forall_{\pi_2}.covers\\
Contains &:= \forall_{\pi_2}.contains\\
CoveredBy &:= \forall_{\pi_1}.coveredBy\\
Inside &:= \forall_{\pi_1}.inside
\end{aligned}$$

For a moving point and a moving region we have just the three basic predicates *Disjoint*, *Meet* and *Inside*, which are defined as above. For two moving points we have the basic predicates *Disjoint* and *Meet*, which are also defined as above. The chosen aggregations are motivated and discussed in detail in [1].

4.3. Developments

Now that we have basic spatio-temporal predicates, the question is how to combine them in order to capture the change of spatial situations. That is, the issue is how to specify *developments*. In order to temporally compose different spatio-temporal predicates, we need a way to restrict the temporal scope of basic spatio-temporal predicates to specific intervals. This can be obtained by *predicate constrictions* (note that $S|_I$ denotes the partial function that yields $S(t)$ for all $t \in I$ and is undefined otherwise): let I be a (half-) open or closed interval. Then

$$P_I := \lambda(S_1, S_2).P(S_1|_I, S_2|_I).$$

Now we can define the *composition* of predicates as follows:

$$P \textbf{ until } p \textbf{ then } Q := \lambda(S_1, S_2).\exists t : p(S_1(t), S_2(t)) \wedge P|_{]-\infty,t[}(S_1, S_2) \wedge Q|_{]t,\infty[}(S_1, S_2)$$

When we now consider how spatial situations can change over time, we observe that certain relationships can be valid 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*. For example, an airplane and a hurricane can meet at a certain instant or for a whole period. Predicates that can only hold on intervals are: *disjoint*, *overlap*, *inside*, *contains*; these are called *period predicates*. For example, it is not possible for an airplane to be disjoint from a hurricane only at one point in time; they have the inherent property to be disjoint for a period.

It is now interesting to see that in satisfiable and uninterrupted developments instant and period predicates always occur in alternating sequence. For example, it is not possible that two continuously changing spatio-temporal objects satisfy *Inside* immediately followed by *Disjoint*. In contrast, *Inside* first followed by *meet* (or *Meet*) and then followed by *Disjoint* can be satisfied. Hence, developments are represented by alternating sequences of spatio-temporal predicates and spatial predicates and are written down by juxtaposition (in this paper). A more formal treatment of compound spatio-temporal predicates and developments is given in [1]. Our example of a flight running into a hurricane can now be formulated as the composition:

Disjoint **until** *meet* **then** *Inside*

Since predicate composition is associative, we can abbreviate nested compositions by writing down simply a sequence of the spatio-temporal and spatial predicates, that is, we can simply write *Disjoint meet Inside* for the above example. We introduce the name Enter for it to reuse it later. A flight running out of a hurricane can be characterized by *Leave* := *Inside meet Disjoint*. A flight that traverses a hurricane can be written as *Disjoint meet Inside meet Disjoint* using basic spatio-temporal predicates or shorter as *Enter Leave* using derived predicates; we introduce the name *Cross* for it. Note that spatial predicates and their corresponding spatio-temporal predicates (like *meet* and *Meet*) that occur next to each other in a development can be merged to the respective spatio-temporal predicate (see also

Section 6). We list a few further examples for two evolving regions:

$$
\begin{aligned}
\mathit{Enter} &:= \mathit{Disjoint\ meet\ Overlap\ coveredBy\ Inside}\\
\mathit{Leave} &:= \mathit{Inside\ coveredBy\ Overlap\ meet\ Disjoint}\\
\mathit{Cross} &:= \mathit{Enter\ Leave}\\
\mathit{Touch} &:= \mathit{Disjoint\ meet\ Disjoint}\\
\mathit{Bypass} &:= \mathit{Disjoint\ meet\ Disjoint}\\
\mathit{Graze} &:= \mathit{Disjoint\ meet\ Overlap\ meet\ Disjoint}
\end{aligned}
$$

In order to assess the expressiveness of our visual notation we can ask which developments are possible at all and which developments can be specified by our visual language. Possible transitions of spatio-temporal relationships can be represented in the so-called *development graphs* whose vertices are labeled either with a spatial, that is, an instant, predicate or with a basic spatio-temporal predicate. Hence, each vertex models a time point or a time interval in which the corresponding predicate is valid. An edge (p, q) represents the transition from a predicate p to a predicate q and stands for $p\,q$. A path $(p_1, p_2, \ldots, p_n)$ within the graph describes a possible temporal development $p_1 p_2 \ldots p_n$ of topological relationships between two spatial objects.

Since due to cycles infinitely many paths of possibly infinite length exist in the development graph, we appropriately restrict the set of possible paths. First, we only consider paths of finite length. Second, we take into account the observation that if a path properly contains an instant predicate together with its corresponding basic spatio-temporal predicate, we can regard this situation as a reoccurrence of a topological relationship. The only difference concerns the temporal duration of the topological relationship. This leads us to the notion of a *quasi-cycle* as a path $v = (v_1, \ldots, v_n)$ in the development graph such that v is a cycle or such that v_1 is a spatial predicate and v_n is its corresponding spatio-temporal predicate, or vice versa. Then a *development path* of a development graph is a path in the graph which does not properly contain any quasi-cycles.

For the point/point and for the point/region case we obtain the following development graphs shown in Fig. 9. Starting, for example, with *Inside* in the point/region case, we obtain seven possible development paths, see Fig. 10.

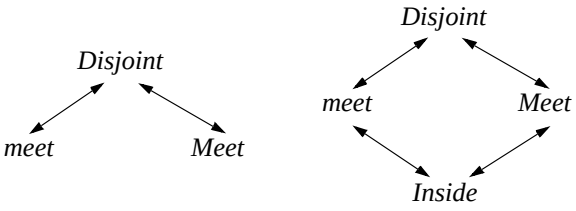

Fig. 9. Development graphs for the point/point and the point/region case.

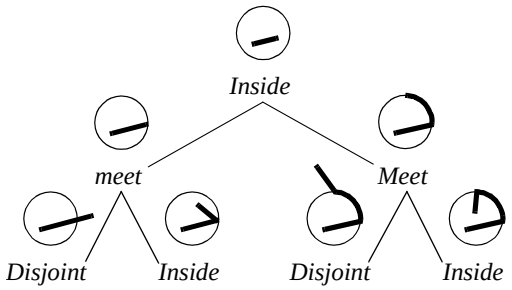

Fig. 10. Examples of development paths.

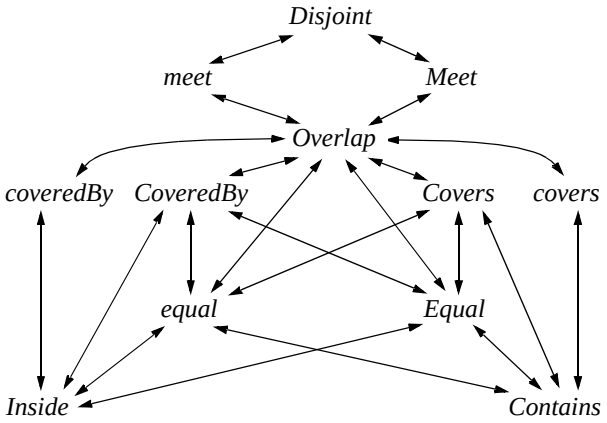

Fig. 11. Region/region development graph.

Since the development graph is symmetric in this case (each of the four vertices can be selected as the start vertex of a path), we obtain a total of 28 paths. This means, there are 28 distinct temporal evolutions of topological changes between a moving point and an evolving region without repetitions. For each alternative we could define an own spatio-temporal predicate. In the point/point case we get 13 possible development paths. The development graph for the region/region case contains 2198 paths and thus possible predicates [1]. It is shown in Fig. 11.

There are some constraints imposed by our visual notation which restrict the possible development paths that can be expressed by a visual specification; consequently, they lead to a restriction of the development graph. These constraints are: (i) the sizes of the static circle and the moved circle are fixed, (ii) the static circle is larger than the moved circle, and (iii) our visual notation contains an implicit ordering of both circles, that is, the smaller moved circle symbolizes always the first argument of a predicate and the larger constant circle stands always for its second argument. These constraints lead to the following restrictions of the development graph.

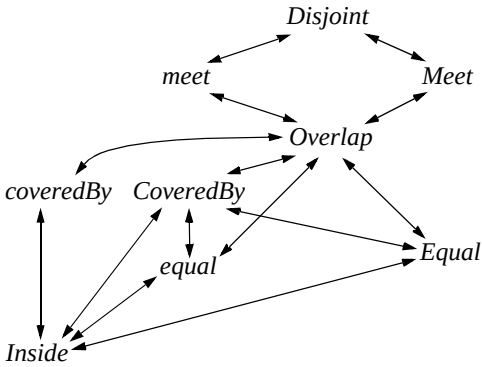

Fig. 12. Removing the predicates *covers*, *Covers* and *Contains* with their incident edges.

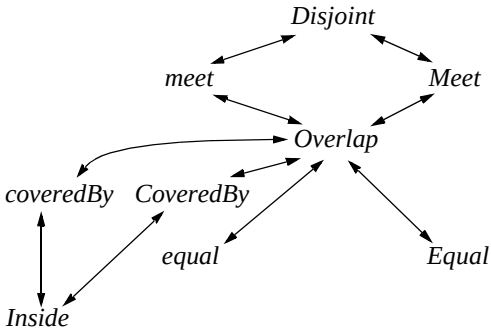

Fig. 13. Removing four edges due to the impossibility of changing an object's extent.

First, from the three pairs *coveredBy*/*covers*, *CoveredBy*/*Covers* and *Inside*/*Contains* only one relationship per pair, namely *coveredBy*, *CoveredBy*, and *Inside*, can be represented in our visual development specifications. Hence, we can remove the vertices *covers*, *Covers*, and *Contains* and their incident edges[1] (see Fig. 12).

Second, four transitions in the graph, namely from *CoveredBy* and from *Inside* to *equal* and *Equal*, respectively, solely result from a growing or shrinking of one object. Since we cannot alter the proportions neither of the static circle nor of the moved smaller circle, the vertices *equal* and *Equal* cannot be reached by *CoveredBy* and *Inside* (and vice versa) so that we can take away the corresponding eight edges (see Fig. 13).

Third, the transitions between *Overlap* and *equal* and between *Overlap* and *Equal* do not require growing or shrinking. But the prerequisite for this transition is that the static circle and the moved circle have the same size, and just this is excluded by

[1] Note that this restriction could be dropped if we would allow that the moved circle can be made larger than the constant circle. (Then the γ-matrix in Section 6 would have to be adapted.)

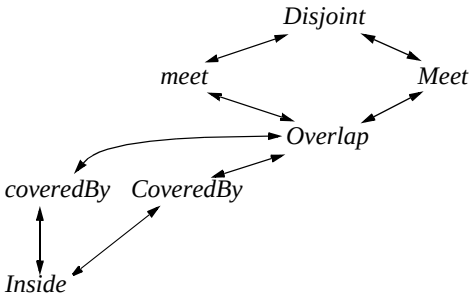

Fig. 14. Removing the predicates *equal* and *Equal* with their incident edges.

our visual notation. From an *Overlap* situation we can never come to an *equal* or an *Equal* situation (and vice versa) so that the four corresponding edges must be removed. Because the vertices *equal* and *Equal* are isolated now, we can remove them, too. We obtain the following final development graph shown in Fig. 14. In this restricted graph, only 87 different development paths are possible.

All finite paths that can be obtained by this development graph (and, of course, all those that can be obtained by concatenation) can be specified with our visual language for the region/region case.

5. Visual specifications of developments

In Section 3, we have described the visual specification mechanism from an application and user interface point of view, that is, how such a query interface could be realized in practice. With the formal background provided by the previous section, this section now gives the reasons and the motivation for our design by discussing a collection of underlying design criteria. For this purpose, we review some of the aspects of Section 3 from a design perspective. The design criteria will eventually lead to a two-dimensional visual language.

The first design decision is essential to obtain an integrated notation for spatial and temporal aspects:

(1) Represent the temporal dimension geometrically. This leads in a first step to a three-dimensional model of spatio-temporal objects.

Now we could stop here and use three-dimensional pictures to specify developments, but in our special application there are two main reasons for not doing so: first, three-dimensional pictures contain much more information than two-dimensional pictures and require therefore generally more input for their generation. This input has to be provided by the user, which means that it can be generally expected that it takes more time and effort to draw three-dimensional than two-dimensional pictures. In particular, without specialized user input devices, it can become quite tedious to generate three-dimensional drawings with mouse and keyboard. Such a drawing interface is also more difficult to implement. Second, three-dimensional illustrations of developments are overdetermined in the sense that

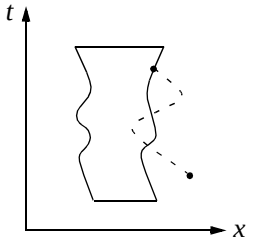

Fig. 15. Simplifying two-dimensional objects.

they display (i) growing/shrinking and movement of regions and (ii) relative positions of the beginnings and endings of objects' lifetimes. (The first point will be discussed in more detail below.) Such overspecifications are generally undesirable since they can complicate the understanding of visual notations: the user has to sort out much visual information that has no meaning for her specification.

The second design decision is essentially a step to reduce overdetermination:

(2) Abstract from exact positions/extents, and reduce two-dimensional geometric objects to one-dimensional ones. Use the y-axis to represent the temporal dimension.

This essentially means to 'forget' about the y-axis with regard to spatial information, and to represent a point at time t as a point on the x-axis and a region as an x-interval. Thus, the y-axis can capture the temporal aspect of spatio-temporal objects so that a moving point is represented by a line and an evolving region is represented by a region as shown in Fig. 15.

This picture describes a moving point that enters a region, then leaves the region and finally stops on the region's border. It is striking that the sketched movement/shrinking/growing of the interval representing the evolving region does not contribute anything to this specification, that is, it would be as well possible to use a plain rectangle representing a stationary/constant region. The reason is that we are only specifying topological relationships, and thus we need only information about the relative positions of objects with respect to each other. In particular, we need not be concerned about the exact position or size information of objects.[2]

This leads to the third design decision:

(3) Represent the evolving region in the definition of a point/region predicate, respectively, one evolving region in the definition of a region/region predicate, simply as a static circle. Likewise, represent one of the two moving points in a point/point spatio-temporal predicate as a static vertical line.

This leads to a uniform notation. For instance, the point/region predicate *Bypass* can be specified as shown in Fig. 16. The trace of the point's movement is interactively specified by the user and depicted by a dotted line.

[2] Actually, this is not the whole truth: for some spatio-temporal developments, growing and shrinking *is* essential, but these cases are rare, and the complexity of an extension of the visual notation would not be justified by the relatively small gain in expressiveness, see Section 4.

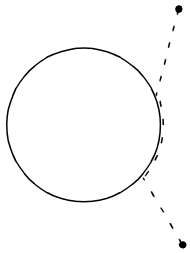

Fig. 16. Visual specification of *Bypass*.

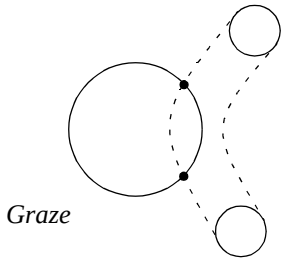

Fig. 17. Visual specification of *Graze*.

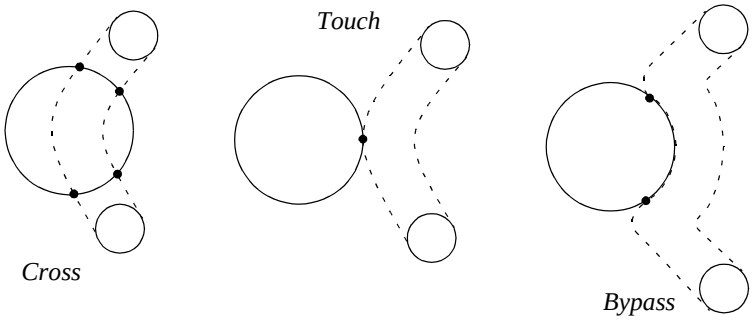

Fig. 18. More spatio-temporal predicates.

In the region/region case, two circles are displayed which show the circle's (evolving region's) first and last position, are smaller than the static circle, and are connected by a trace specifying the circle's movement. The trace itself is depicted by two dotted lines. For example, the predicate *Graze* is drawn as shown in Fig. 17.

The trace represents the sequence *Disjoint meet Overlap meet Disjoint*. This is because the left trace border intersects the constant region in exactly two points and the right trace border does not intersect the constant region at all. Hence, altogether this picture denotes the predicate *Graze*. Some variations are shown in Fig. 18.

Note that the exact interpretation can always be inferred from the intersections of the trace borders with the static circle. This will be explained in the next section.

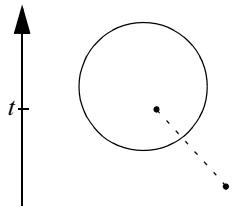

Fig. 19. Intermediate specification.

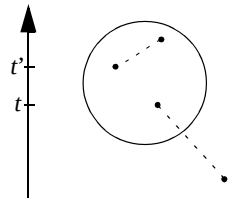

Fig. 20. Continuation of specification: a gap.

So far we have only considered continuously moving objects and the spatio-temporal predicates that can be derived from them when interacting with a static object. But, as we have already seen in Section 3, we can well use the described notation with discontinuous objects and even with objects having gaps in their domain. All these cases have a precisely defined correspondence in the user interface. Consider, for instance, a moving point that has entered the constant circle. When the user releases the mouse button, say, at y-position or *time*-value t, this specifies the spatio-temporal predicate *Disjoint meet Inside*, see Fig. 19.

Now there are two ways to continue this specification:

(1) The line is continued from the last position.
(2) A new line is added with an initial point that has the a *time*-value t' that is greater than t.

This gap in the drawn object means there is some time between t and t' which we do not care about, that is, the relationship between the moving object and the static object during this period is not to be specified but rather arbitrary. In this case we do not have to restrict the new position, in particular, it may well specify the same spatial predicate as the last position of the preceding movement. In our example it is therefore possible to continue (and complete) a movement inside the circle (see Fig. 20). This denotes the development *Disjoint meet Inside True Inside*. The inserted *True* predicate serves as a kind of wildcard predicate: between t and t' anything can happen, but after t', the point has to stay inside the circle. Hence, points that just enter the static object satisfy this predicate as well as points that enter and after that touch or leave the static object many times if they finally stay inside.

Let us finally note that the visual notation is also well suited to express predicates on three or more spatio-temporal objects. Consider, for example, the task of finding

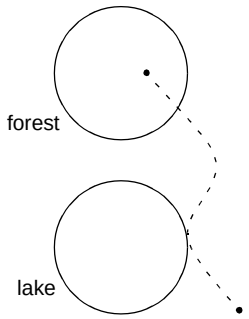

Fig. 21. Predicates on more than two objects.

out animals that have visited a lake and then entered a forest. This can be easily expressed by the specification shown in Fig. 21.

It is the fact that the temporal evolution of constant regions is ignored in trace pictures that makes the spatio-temporal relationships between *forest* and *lake* irrelevant; and the use of the y-axis for representing the temporal dimension expresses in the above picture that first the lake has to be visited and after that the forest has to be entered. (By flipping the picture upside down, a query asking for animals leaving the forest and then touching the lake would be expressed.)

6. Interpretation of development specifications

In this section we describe how sequences of spatio-temporal predicates are inferred from visual development specifications. In the following, we refer to the constant and to the moving object of a specification. The constant object can be either a vertical line or a circle. If the moving object is a moving point, it is shown as a curve representing the trace, otherwise it is given by two circles showing the first and last position of the moving region, and two parallel curves representing the borders of the trace. Note that all curves are monotonic with respect to the y-axis (that is, t-axis) because an object cannot move backward in time. Moreover, the two curves representing the borders of a moving region's trace do not intersect. We denote these two lines by l_1 and l_2. (Note that the naming is arbitrary because the γ-matrix (see below) that defines the computation of spatio-temporal predicates is symmetric.)

Next, we compute the intersection *IS* of the constant object and the trace representation of the moving object, that is, one line in case of a moving point and two lines in case of a moving region. *IS* contains a t-sorted sequence of predicates for the resulting line segments and intersection points and for the begin and end points of the trace. In the region/region case *IS* consists of two such sequences. More precisely, *IS* contains pairs (t, P) where t denotes a value of the t-axis and P gives the relationship of the point or line segment with respect to the constant object. Actually, *IS* contains for each but the last time value two tuples: one for the spatial

predicate that holds for the point at t and one for the spatial predicate that holds for the line segment starting from t. The spatial predicate for the line segment will later be interpreted as a basic spatio-temporal predicate that holds for the point during its movement from t up to the next time value.

With this information we are almost done, at least in the case of point/point and point/region specifications: we can simply take the sequence of predicates from IS. Although there might be some surplus instant predicates, these do not affect the semantics of the resulting development predicate. For example, extracting the predicates from IS for Fig. 16 gives (predicates with an exponent L are line predicates):

$$disjoint\ disjoint^L\ meet\ meet^L\ meet\ disjoint^L\ disjoint$$

Interpreting the line predicates, such as $disjoint^L$, as spatio-temporal predicates yields the development:

$$disjoint\ Disjoint\ meet\ Meet\ meet\ Disjoint\ disjoint$$

Actually, this denotes the same development as

$$Disjoint\ Meet\ Disjoint$$

This is because Lemma 1 allows the simplification of predicate sequences:

Lemma 1. *Let q/p/i be an arbitrary/period/instant spatial predicate, and let Q/P/I be the corresponding spatio-temporal predicate. Then*:

(a) $qQ = Q$
(b) $IiP = IP$
(c) $\forall o, o' : dom(o) \bigcap dom(o')\ is\ right - closed \Rightarrow (Qq)(o, o') = Q(o, o')$
(d) $QQ = Q$.

We can use this lemma to repeatedly rewrite a predicate sequence to a short, non-redundant normal form. For the above example, we can reduce with part (a) *disjoint Disjoint* to *Disjoint* and *meet Meet* to *Meet*, and with part (b) we can replace the sequence *Meet meet Disjoint by Meet Disjoint*. Finally, we can apply part (c) to reduce *Disjoint disjoint* to *Disjoint*.

A bit more complex is the treatment of the region/region case. First, intersection yields two sequences IS_1 and IS_2 of predicates, for the two trace lines l_1 and l_2, respectively. These have to be scanned in parallel to build a new predicate sequence $\bar{P}$. Essentially, in each step one predicate from IS_1 is combined with one predicate from IS_2 into a new predicate. We do that as follows: first, we compute a sorted list T of all t-values from both sequences. T describes the events at which a predicate in either sequence changes. The elements of T contain first the two starting points of the two trace lines, followed by all intersection points, and finally the end points of the trace lines. Then we inspect IS_1 and IS_2 for each $t \in T$. With $IS_i[t]$ we denote the predicate that holds at time t in the sequence IS_i. If IS_i contains a pair for t (recall that for each but the last time point, it will actually contain two pairs), the pair with the instant predicate is selected. Otherwise, a period predicate is selected for the

Table 1

γ	$disjoint^L$	$disjoint$	$meet^L$	$meet$	$inside^L$	$inside$	$undef$
$disjoint^L$	*Disjoint*	*disjoint*	*Meet*	*meet*	*Overlap*	*overlap*	*True*
$disjoint$		*disjoint*	*meet*	*meet*	*overlap*	*overlap*	*True*
$meet^L$			*Meet*	*meet*	*CoveredBy*	*coveredBy*	*True*
$meet$				*meet*	*coveredBy*	*coveredBy*	*True*
$inside^L$					*Inside*	*inside*	*True*
$inside$						*inside*	*True*

largest t' in IS_i that is smaller than t (because this predicate holds until the next time point in IS_i). It is clear that in each step either IS_1 or IS_2 contributes an instant predicate because one sequence has contributed its time value to T. With $IS_i[t^>]$ we denote the period predicate that holds immediately after time t in the sequence IS_i. This is given by the tuple (t', P) for the largest t' in IS_i with $t' \leq t$ (where P is a spatio-temporal predicate). It is striking that, in general, the time intervals spanned by IS_1 and IS_2, given by the minimum and maximum contained t-value, do not agree. Thus, for some $t \in T$, $IS_i[t]$ will be undefined, and we define $IS_i[t]$ at those places to yield the dummy predicate *undef*. This simplifies the definition of the combination of predicates described next.

The combination of two predicates is defined by a function γ which is defined in Table 1: the row and column headers show the spatial predicates for points and lines, and the table fields give the corresponding derived spatio-temporal predicates. Table 1 defines a function that maps a pair of two-dimensional spatial predicates that can hold for a point/line and a circle (where the predicates for lines are written with an exponent L) to a spatial or a basic spatio-temporal predicate for two evolving regions. Note that the table is symmetric (that is, γ is commutative); we have therefore omitted the lower left part for clarity. Table 1 is used as follows: for each time value t we look up the table entry for row $IS_1[t]$ and column $IS_2[t]$ to get the spatial or spatio-temporal predicate that holds at time t for the two evolving regions. Similarly, we look up the table entry for row $IS_1[t^>]$ and column $IS_2[t^>]$ to get the spatio-temporal predicate that holds immediately after t for the two evolving regions.

Before we proceed, let us illustrate the above definitions with an example. Consider the visualization of the predicate *Cross* from Fig. 18. Here we show the time points for any change in the predicate sequences, see Fig. 22: we get 4 intersection time points $t_1, \ldots, t_4$ (sorted by increasing t-coordinate): t_1 and t_4 result from l_1, and t_2 and t_3 result from l_2, and we have two start points (t_a for l_1 and t_b for l_2) and two end points (t_y for l_2 and t_z for l_1):

With these time points, we obtain the following predicate sequences:

$$IS_1 = [(t_a, disjoint), (t_a, disjoint^L), (t_1, meet), (t_1, inside^L), (t_4, meet), (t_4, disjoint^L), (t_z, disjoint)]$$

$$IS_2 = [(t_b, disjoint), (t_b, disjoint^L), (t_2, meet), (t_2, inside^L), (t_3, meet), (t_3, disjoint^L), (t_y, disjoint)]$$

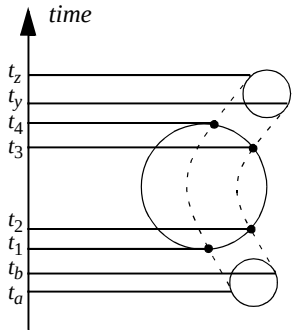

Fig. 22. Events in the visualization of the Cross predicate.

Obviously, we have $T = [t_a, t_b, t_1, t_2, t_3, t_4, t_y, t_z]$, and the values for $IS_i[t_j]$ and $IS_i[\overrightarrow{t_j}]$ for $i \in \{1, 2\}$ and $j \in \{a, b, 1, 2, 3, 4, y, z\}$ are shown in the following table:

t	t_a	$\overrightarrow{t_a}$	t_b	$\overrightarrow{t_b}$	t_1	$\overrightarrow{t_1}$	t_2	$\overrightarrow{t_2}$	t_3	$\overrightarrow{t_3}$	t_4	$\overrightarrow{t_4}$	t_y	$\overrightarrow{t_y}$	t_z
$IS_1[t]$	*disjoint*	*disjoint*^{L}	*disjoint*^{L}	*disjoint*^{L}	*meet*	*inside*^{L}	*inside*^{L}	*inside*^{L}	*inside*^{L}	*inside*^{L}	*meet*	*disjoint*^{L}	*disjoint*^{L}	*disjoint*^{L}	*disjoint*
$IS_2[t]$	*True*	*True*	*disjoint*	*disjoint*^{L}	*disjoint*^{L}	*disjoint*^{L}	*meet*	*inside*^{L}	*meet*	*disjoint*^{L}	*disjoint*^{L}	*disjoint*^{L}	*disjoint*	*True*	*True*

Next we can compute the derived predicates with the help of γ. Let us explain some of the entries in the γ-table: for time points t_a, $\overrightarrow{t_a}$, $\overrightarrow{t_y}$ and t_z we obtain the unit predicate *True*. Whenever both predicates agree, a predicate of the same name is computed, and this is a spatio-temporal predicate only if both predicates are period predicates. We therefore obtain *Disjoint* at time points $\overrightarrow{t_b}$ and $\overrightarrow{t_4}$, *disjoint* for t_b and t_y, and *Inside* at $\overrightarrow{t_2}$. When one side of the moving circle meets and the other side is at the same time inside, this means that the moving circle is covered by the static circle. Similarly, if one side meets and the other side is disjoint, this is the case if both circles touch. Hence, we get *coveredBy* at t_2 and t_3 and *meet* at t_1 and t_4. Finally, if one side is disjoint and the other side is inside, this can only be the case if both circles overlap. This means that γ yields *Overlap* for $\overrightarrow{t_1}$ and $\overrightarrow{t_3}$. Altogether, we obtain the following sequence of predicates:

t	t_a	$\overrightarrow{t_a}$	t_b	$\overrightarrow{t_b}$	t_1	$\overrightarrow{t_1}$	t_2	$\overrightarrow{t_2}$	t_3	$\overrightarrow{t_3}$	t_4	$\overrightarrow{t_4}$	t_y	$\overrightarrow{t_y}$	t_z
$\gamma(IS_1[t], IS_2[t])$	*True*	*True*	*Disjoint*	*Disjoint*	*Meet*	*Overlap*	*coveredBy*	*Inside*	*coveredBy*	*Overlap*	*meet*	*Disjoint*	*disjoint*	*True*	*True*

There are two further points to note before we can describe the complete algorithm to compute $\bar{P}$.

First, the predicates obtained from the trace intersections are not sufficient to derive the denoted development predicate. Consider, for instance, the situation shown in Fig. 23.

Obviously, this specifies the development *Disjoint meet*, but both trace lines contain only predicates *disjoint* and *Disjoint* (we have indicated the endpoints of the trace lines), and thus *meet* cannot be inferred. Therefore, we have to compute the spatial predicates that hold for the initial and final circle, too, and join these

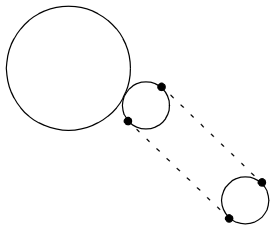

Fig. 23. Trace intersections are not sufficient.

predicates at both ends of the predicate sequence $\bar{P}$. For the above picture we obtain *disjoint* and *meet*, and for the *Cross* example we get two times *disjoint*. We denote these predicates by p_I and p_F.

The second observation is related to the previous point: the spatial predicate obtained for an initial or final circle can be either a period predicate (*disjoint*, *inside*, *overlap*) or an instant predicate (*meet*, *coveredBy*). In the former case, we know that the development will contain the corresponding spatio-temporal predicate. In the latter case it can be either the spatial predicate itself or its spatio-temporal counterpart. Fortunately, we can discriminate between these two cases by looking at the first line predicate in either of the two intersection sequences (to deal with the initial circle): if this predicate is the spatio-temporal predicate that corresponds to the spatial predicate obtained for the initial circle, then this spatio-temporal predicate is taken into the development, otherwise the simple spatial predicate remains. The case for the final circle is analogous: we just look at the last line predicate in either of the two intersection sequences.

Now we can compute the predicate sequence $\bar{P}$ with the algorithm shown below. We use the following notations: $period(p) :\Leftrightarrow p \in \{disjoint, inside, overlap\}$, and P_I and P_F denote the spatio-temporal predicates corresponding to p_I and p_F, respectively. The concatenation of an element and a sequence or of two sequences is performed by the function *concat*, the ith element from a sequence S is denoted by $S[i]$, and the length of a sequence S is given by $|S|$. Finally, the last time value of T requires special treatment. We therefore denote the last element of T by $last(T)$ and the initial part of T, that is, T without $last(T)$, by $init(T)$. (We therefore have: $T = concat(init(T), last(T))$.)

Algorithm InferPredicates
begin
  $\bar{P} := [\,]$
  for each $t \in init(T)$ **do**
    $\bar{P} := concat(\bar{P}, [\gamma(IS_1[t], IS_2[t]), \gamma(IS_1[t^{\succ}], IS_2[t^{\succ}])]$
  $\bar{P} := concat(\bar{P}, \gamma(IS_1[last(T)], IS_2[last(T)]$
  Simplify $\bar{P}$ (by Lemma 1)
  Determine p_I and p_F
  if $(period(p_I) \vee \exists i : IS_i[T[1]^{\succ}] = P_I)$ **then**
    $\bar{P} := concat(P_I, \bar{P})$

else
$\bar{P} \coloneqq concat(p_I, \bar{P})$
if $(period(p_F) \vee \exists i : IS_i[T[|T|-1]^{>}] = P_F)$ **then**
$\bar{P} \coloneqq concat(\bar{P}, P_F)$
else
$\bar{P} \coloneqq concat(\bar{P}, p_F)$
Simplify $\bar{P}$ again
end

We demonstrate the application of the algorithm with the *Cross* example. We have already determined T and the values of γ at the different times. Thus, just before the first simplification we have:

$\bar{P} = [True, True, disjoint, Disjoint, meet, Overlap, coveredBy, Inside, coveredBy, Overlap, meet, Disjoint, disjoint, True, True]$

Simplification then yields:

$\bar{P} = [Disjoint, meet, Overlap, coveredBy, Inside, coveredBy, Overlap, meet, Disjoint]$

Next we obtain *disjoint* for both p_I and p_F. The next two **if**-statements try to fuse these predicates with $\bar{P}$. The first element of T is t_a, and since $IS_1[t_a^{>}] = Disjoint$ we add *Disjoint* to the front of $\bar{P}$. Similarly, the second-last element of $T(= T[|T|-1])$ is t_y, and since $IS_1[t_y^{>}] = Disjoint$, we add *Disjoint* to the rear of $\bar{P}$. Actually, this does not have any effect since the following simplification eliminates [here: Lemma 1 (d)] both predicates. Thus, the development computed for the *Cross* picture is finally as expected:

Disjoint meet Overlap coveredBy Inside coveredBy Overlap meet Disjoint

So far, we have described the inference of predicates only for continuous movements. Fortunately, the method can be also applied to non-continuous movements and also to objects with disconnected domains: first, determine with the above algorithm the predicate sequence for each continuous part of the movement. Then concatenate the predicate sequences, and if the *time*-value of the start circle/point of the $n+1$st movement is not equal to the *time*-value of the end circle/point of the nth movement (then a gap in the domain of the moving object is specified), insert a *True* predicate. Otherwise, that is, when the *time*-values agree (then just a non-continuous movement is specified), drop a possible final spatial predicate in the sequence for the nth movement because the object semantics defines that the object's value is given by the initial value of the $n+1$st movement, and therefore also the corresponding predicate holds.

We illustrate this by a further example. Consider the specification in Fig. 24.

For the lower part we obtain the following predicates of the trace lines:

t	t_a	$t_a^{>}$	t_b	$t_b^{>}$	t_1	$t_1^{>}$	t_2	$t_2^{>}$	t_y	$t_y^{>}$	t_z
$IS_1[t]$	$disjoint$	$disjoint^L$	$disjoint^L$	$disjoint_L$	$meet$	$inside^L$	$inside^L$	$inside^L$	$inside$	$True$	$True$
$IS_2[t]$	$True$	$True$	$disjoint$	$disjoint^L$	$disjoint^L$	$disjoint^L$	$meet$	$inside^L$	$inside^L$	$inside^L$	$inside$

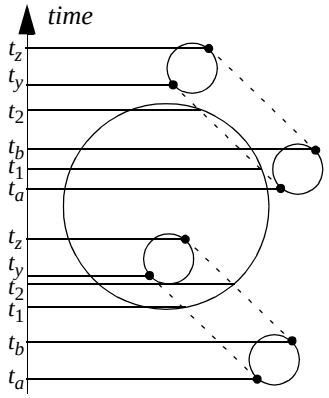

Fig. 24. Specification of a gap.

Using the γ-matrix we obtain the following sequence of predicates:

	t_a	$t_a^>$	t_b	$t_b^>$	t_1	$t_1^>$	t_2	$t_2^>$	t_y	$t_y^>$	t_z
γ	*True*	*True*	*disjoint*	*Disjoint*	*meet*	*Overlap*	*coveredBy*	*Inside*	*Inside*	*True*	*True*

Simplification then yields the development:

$$\bar{P}_l = [Disjoint, meet, Overlap, coveredBy, Inside]$$

For the upper part we obtain the following predicates of the trace lines. Note that irrespective of the spatial ordering in the picture t_a and t_b give always the minimum time-values, which means that, despite actual spatial positions shown in the upper part of Fig. 24, we have $t_b < t_1$.

t	t_a	$t_a^>$	t_b	$t_b^>$	t_1	$t_1^>$	t_2	$t_2^>$	t_y	$t_y^>$	t_z
$IS_1[t]$	*disjoint*	*disjoint*^{L}	*disjoint*^{L}	*disjoint*^{L}	*meet*	*inside*^{L}	*meet*	*disjoint*^{L}	*disjoint*	*True*	*True*
$IS_2[t]$	*True*	*True*	*disjoint*	*disjoint*^{L}	*disjoint*^{L}	*disjoint*^{L}	*disjoint*^{L}	*disjoint*^{L}	*disjoint*^{L}	*Disjoint*^{L}	*disjoint*

Using the γ-matrix we obtain the following sequence of predicates:

t	t_a	$t_a^>$	t_b	$t_b^>$	t_1	$t_1^>$	t_2	$t_2^>$	t_y	$t_y^>$	t_z
γ	*True*	*True*	*disjoint*	*Disjoint*	*meet*	*Overlap*	*meet*	*Disjoint*	*disjoint*	*True*	*True*

Simplification then yields the development:

$$\bar{P}_u = [Disjoint, meet, Overlap, meet, Disjoint]$$

Joining $\bar{P}_l$ and $\bar{P}_u$ requires the insertion of a *True* predicate and finally yields the development specification:

$\bar{P} = [Disjoint, meet, Overlap, coveredBy, Inside, True, Disjoint, meet, Overlap, meet, Disjoint]$

which is the same as *Enter True Graze*.

Note that the described technique can be used in a syntax directed editor as well as in an off-line environment.

7. Conclusions

We have demonstrated how a simple two-dimensional visual language can be used to express topological predicates on spatio-temporal objects. This language can be well used as a query interface to spatio-temporal databases. Having a precise semantics, the visual notation can also serve as a formal language to communicate and reason about spatio-temporal situations in general.

References

[1] M. Erwig, M. Schneider, Spatio-temporal predicates, IEEE Transaction on Knowledge and Data Engineering 14 (2002) 881–901.
[2] M.J. Egenhofer, R.D. Franzosa, Point-set topological spatial relations. *International Journal of Geographical Information Systems*, (1991) 161–174.
[3] M. Erwig, M. Schneider, Visual specification of spatio-temporal developments. *15th IEEE Symposium on Visual Languages*, Tokyo, Japan, 1999, pp. 187–188.
[4] M. Erwig, M. Schneider, Query-by-trace: visual predicate specification in spatio-temporal databases, in: H. Arisawa, T. Catarci (Eds.), Advances in Visual Information Management—Visual Database Systems, Kluwer Academic Publishers, 2002, pp. 199–218.
[5] M. Erwig, M. Schneider, Developments in spatio-temporal query languages. *IEEE International Workshop on Spatio-Temporal Data Models and Languages*, Florence, Italy, 1999, pp. 441–449.
[6] M. Erwig, B. Meyer, Heterogeneous visual languages — integrating visual and textual programming. *IEEE Symposium on Visual Languages*, Darmstadt, Germany, 1995, pp. 318–325.
[7] M.F. Worboys, A unified model for spatial and temporal information, The Computer Journal 37 (1994) 25–34.
[8] S.K. Gadia, S.S. Nair, Temporal databases: a prelude to parametric data. In: A.U. Tansel *et al.* (eds.), *Temporal Databases: Theory, Design, and Implementation,* 1999, pp. 28–66.
[9] T.S. Cheng, S.K. Gadia, A pattern matching language for spatio-temporal databases. *ACM Conference on Information and Knowledge Management*, 1994, pp. 288–295.
[10] M.H. Böhlen, C.S. Jensen, B. Skjellaug, Spatio-temporal database support for legacy applications, *ACM Symposium on Applied Computing*, 1998, pp. 226–234.
[11] M. Erwig, R.H. Güting, M. Schneider, M. Vazirgiannis, Abstract and discrete modeling of spatio-temporal data types, *6th ACM Symposium on Geographic Information Systems*, Washington, DC, USA, 1998, pp. 131–136.
[12] M. Erwig, R.H. Güting, M. Schneider, M. Vazirgiannis, Spatio-temporal data types: an approach to modeling and querying moving objects in databases, GeoInformatica 3 (1999) 269–296.
[13] M. Stonebraker, B. Rubenstein, A. Guttman, Application of abstract data types and abstract indices to CAD data bases, *ACM/IEEE Conference on Engineering Design Applications*, 1983, pp. 107–113.
[14] M. Stonebraker, Inclusion of new types in relational database systems, *International Conference on Data Engineering*, 1986, pp. 262–269.

[15] M. Erwig, M. Schneider, R.H. Güting, Temporal objects for spatio-temporal data models and a comparison of their representations, *International Workshop on Advances in Database Technologies*, Lecture Notes in Computer Science, 1552, 1998, pp. 454–465.
[16] T.S. Yeh, B. de Cambray, Time as a geometric dimension for modeling the evolution of entities: a 3D approach, *International Conference on Integrating GIS and Environmental Modeling*, 1993.
[17] T.S. Yeh, B. de Cambray, Modeling highly variable spatio-temporal data, *Sixth AustraliAsian Database Conference*, 1995, pp. 221–230.
[18] A. Segev, A. Shoshani, A temporal data model based on time sequences, in: A. U. Tansel et al. (Eds.), *Temporal Databases: Theory, Design, and Implementation*, 1993, pp. 248–270.
[19] R.H. Güting, M.H. Böhlen, M. Erwig, C.S. Jenssen, N.A. Lorentzos, M. Schneider, M. Vazirgiannis, A foundation for representing and querying moving objects, ACM Transactions on Database Systems 25 (2000) 1–42.
[20] J.F. Allen, Towards a general theory of action and time, Artificial Intelligence 23 (1984) 123–154.
[21] A. Galton, Towards a qualitative theory of movement. *International Conference on Spatial Information Theory*, Lecture Notes on Computer Science, Vol. 988, 1995, pp. 377–396.
[22] Z. Cui, A.G. Cohn, D.A. Randell, Qualitative and topological relationships in spatial databases, *International Symposium on Advances in Spatial Databases*, Lecture Notes in Computer Science, Vol. 692, 1993, pp. 296–315.
[23] M.A. Aufaure-Portier, Ahigh level interface language for GIS, Journal of Visual Languages and Computing 6 (1995) 167–182.
[24] D. Calcinelli, M. Mainguenaud, Cigales, a visual query language for a geographical information system: the user interface, Journal of Visual Languages and Computing 5 (1994) 113–132.
[25] M.J. Egenhofer, Spatial-query-by-sketch. *IEEE Symposium on Visual Languages*, Boulder, CO, USA, 1996, pp. 60–67.
[26] M. Hirakawa, E. Jungert, An image database system facilitating icon-driven spatial information definition and retrieval. *IEEE Symposium on Visual Languages*, 1991, pp. 192–198.
[27] Y.C. Lee, F.L. Chin, An iconic query language for topological relationships in GIS, Journal of Visual Languages and Computing 9 (1995) 25–46.
[28] B. Meyer, Pictorial deduction in spatial information systems. *IEEE Symposium on Visual Languages*, Seattle, WA, USA, 1994, pp. 23–30.
[29] M. Wessel, V. Haarslev, VISCO: bringing visual spatial querying to reality. *IEEE Symposium on Visual Languages*, Halifax, Nova Scotia, Canada, 1998, pp. 170–177.
[30] T. Arndt, S.K. Chang, Image sequence compression by iconic indexing. *IEEE Workshop on Visual Languages*, 1989, pp. 177–182.
[31] A. Del Bimbo, E. Vicario, D. Zingoni, Symbolic description and visual querying of image sequences using spatio-temporal logic, IEEE Transactions on Knowledge and Data Engineering 7 (1995) 609–621.
[32] I.M. Walter, R. Sturm, P.C. Lockemann, A semantic network based deductive database system for image sequence evaluation. *Second IFIP Working Conference on Visual Database Systems*, 1992, pp. 251–276.