DESIGNING OPERATORS FOR AN OBJECT-ORIENTED SPATIO-TEMPORAL DATA MODEL

Ale RAZA[*] and Wolfgang KAINZ[**]
Geoinformatics, Spatial Information Theory and Applied Computer Science
International Institute for Aerospace Survey and Earth Sciences (ITC)
PO Box 6, 7500 AA Enschede, The Netherlands.
Tel: +31 – 53 – 4 874 374
[*]saraza@itc.nl, [**] kainz@itc.nl

Working Group TC IV-4

KEY WORDS: Object-orientated, Spatio-temporal, Data model, Design, Topology.

ABSTRACT

A spatio-temporal data model consists of data structure, operations and consistency rules. Designing and implementing such model has been a challenging task because of its complexity. This paper focuses on operators for a spatio-temporal model. Two types of operators can be contemplated in this model, i.e., static and dynamic operators. Dynamic operators such as create, kill, destroy and reincarnate, which change the system status are identified. These operators are based on author's previous work on an object oriented approach to design a unified cell tuple based spatio-temporal data model. Unified Modeling Language (UML) and Object Constraints Language (OCL) are used to design this model. Operations governing the SpatioTemporalClass of this model are discussed here. SpatioTemporalClass is a super-class of ZeroTCellClass, OneTCellClass and TwoTCellClass. Topological notions of point set approach are employed to analyze the various intersections of objects of SpatioTemporalClass. It is demonstrated how the create and kill operators for ZeroTCellClass and OneTCellClass can be constructed in a consistent fashion. Designing these operators may pave the way to design and implement a generic temporal GIS.

1 INTRODUCTION

One of the fundamental enigmas and impediments in designing a generic temporal GIS (TGIS) is the spatio-temporal data model. To design such a model has been a challenging task for many researchers because of the complex data structure. Few integrated approaches exist that treat spatio-temporal data in a unified fashion. Employing object-oriented (OO) design and relying on a solid mathematical basis may reduce the complexity of the data structure. Object-oriented design may facilitate to model space, time and attribute (three components of real world feature objects) in a modular and systematic manner. The OO technique is a natural way to design a model, and its importance in spatial and temporal databases has been acknowledged by many researchers. This may provide a cleaner data model, where each component may be defined in its own and may be integrated in a more structured and flexible mode. The mathematical concepts provide a sound basis to define the space and time, which may provide an unambiguous definition of data model for implementation and a basis for further development of a spatio-temporal query language. The data model consists of data structure, operations, and consistency rules. A class is a set of objects with common properties. A class consists of data members and member function (operations). The former is called static and latter dynamic aspects of the class. The structure of the data model is defined by data members (data), while operations and consistency rules are defined by operations. This paper focuses on the dynamic part of the class, i.e., member functions or operators, which is based on the authors previous work on a unified OO approach to design a spatio-temporal data model (Raza and Kainz, 1999). The earlier work presents the data members and consistency rules. Also various classes were defined. SpatialClass, TemporalClass and AttributeClass represent the space, time and attribute component of a real world feature object, respectively. SpatioTemporalClass and AttributeTemporalClass are the aggregation of Spatial-, Temporal- and Attribute-Classes. This cell tuple based spatio-temporal data model is briefly introduced in section-2.2. This paper focuses on operations governing the two subclasses (ZeroTCell and OneTCell) of SpatioTemporalClass to construct this model. These operators have been implemented in Visual C++. Methods are the implementation of the operation and specify the algorithm associated with the operation. Algorithms for spatio-temporal databases are different from spatial databases. These algorithms are not discussed in this paper. An OO visual-modeling language, i.e., Unified Modeling Language (UML), is used to define the data structure and operations. First UML is based on solid semantics and notation definition, which is necessary for interoperability. Secondly, UML is an evolution from Booch (Booch, 1994) object modelling technique (OMT), OO software engineering (OOSE) and other OO methods, which have proven tracks of successful implementation in complex software design. It supports the definition of interfaces to objects as data (attributes), operations and association. Operations of SpatioTemporalClass are defined in

UML in compliance with the UML notation guide. To further elaborate and to avoid the unambiguous specification Object Constraints Language (OCL) is used. An OCL is a formal, pure expressive language that is easy to read and write. It has been developed by IBM Insurance Division (UML 1.3, 1999).

2 UNIFIED CELL TUPLE BASED SPATIO-TEMPORAL DATA MODEL

The conceptual schema for the spatio-temporal data model is presented here. UML is employed to design a unified spatio-temporal data model. First we briefly introduce the UML presentation, notation and organization followed by cell tuple based spatio-temporal data model.

2.1 Unified Modeling Language (UML)

The vocabulary used in the design of the model is presented in Figure 1. They are based on UML notations. The constraints are defined in Object Constraints Language. Following conventions are used to define the data members and member functions (operations) of the classes. They are in compliance with UML notation guide (UML 1.3, 1999).

Data members	:	<<stereotype>> visibility name: type-expression = initial-value
Operations	:	<<stereotype>> visibility name (parameter-list): return-type-expression

parameter-list	:	comma separated list of formal parameters with following syntax.
kind name	:	type-expression = default-value
<<stereotype>>	:	Tag for data member (attribute) or operations.
visibility	:	+ public visibility
	:	- private visibilty
	:	# protected visibility
name	:	identifier string that represents the name of the attribute (data type).
type-expression	:	language-dependent specification of the implementation of the attribute.
initial-value	:	initial value of newly created object.
kind	:	in, out, or inout (default is in)
name	:	name of formal parameter
default-value	:	optional value expression for parameter.

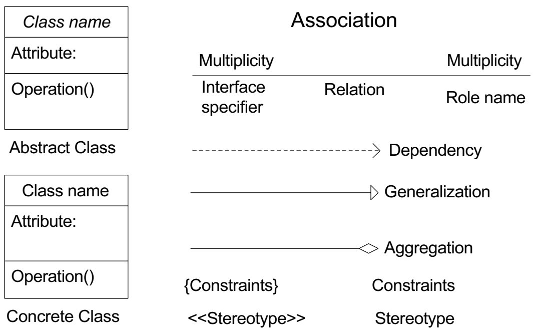


Figure 1: Icons used to represent the model.

The abstract class, attribute or operations are defined in *italic*. In addition to UML types, stereotype can be defined in UML to add extra semantics. They are defined within guillemets <<stereotype >>. Following are the list of stereotypes use in designing the model.

<<index>>	:	Information can be used in indexing.
<<time>>	:	The object is defined in time. This is assumes as a linear time.
<<dynamic>>	:	The kind of operation create, kill, destroyed or reincarnate (discussed later). This operation may modify the state of the system.
<<query>>	:	The nature of operation is query. This operation does not modify the system's state.

Following data type corresponds to the collection. The type collection is predefined in OCL. There are three collection types in OCL, i.e., Set, Bag, and Sequence. Set is the mathematical set. A Bag is like set. A sequence is like a bag in which the elements are ordered. Collection is a set as defined in the OCL and can be of following types.

Set{ }	:	a normal set. Members or objects of the set appeared only one and order is not important.
Sequence{ }	:	an ordered set, with no repetition.
Bag{ }	:	A collection of object, where repetition is allowed.

2.2 Conceptual Schema

The three components of reality, i.e., space, time and attribute form three classes, SpatialClass, TemporalClass, and AttributeClass (Figure 2). These classes are aggregated to create the SpatioTemporalAttribute-Class, commonly known as FeatureClass. The three classes, which are aggregated to generate the SpatioTemporalAttributeClass provide a basis for modeling STAO, which is the backbone of any TGIS. SpatialClass has one PointClass (subclass). A SpatioTemporalClass is a superclass of three classes, i.e., ZeroTCellClass, OneTCellClass and TwoTCellClass. The objects of ZeroTCellClass, OneTCellClass and TwoTCellClass are temporal- node, arc and polygon, respectively.

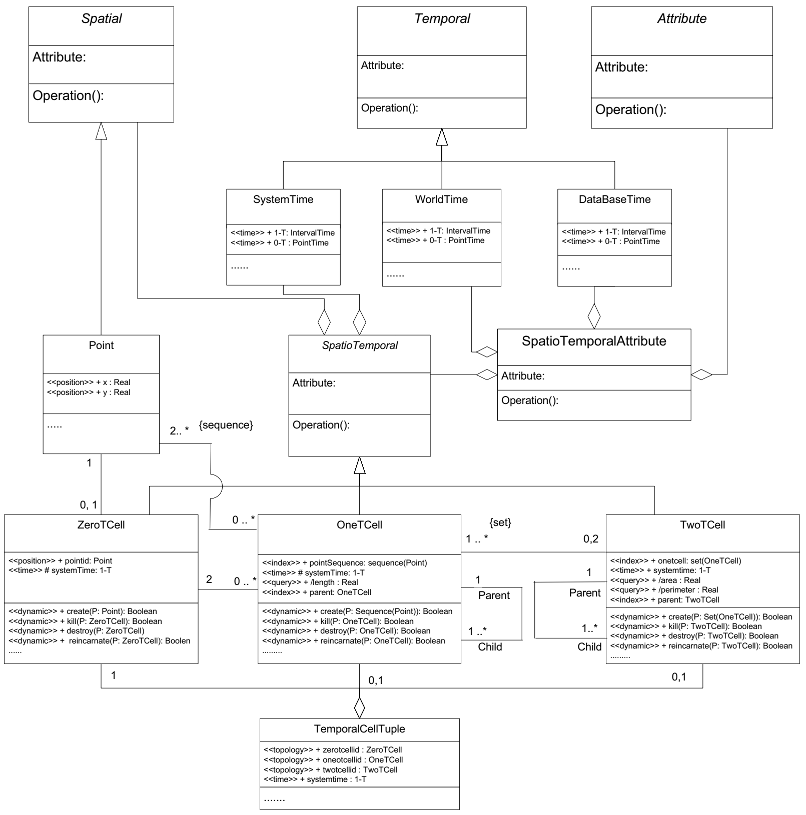

Spatial
Attribute:
Operation():
Temporal
Attribute:
Operation():
Attribute
Attribute:
Operation():
SystemTime
<<time>> + 1-T: IntervalTime
<<time>> + 0-T : PointTime
......
WorldTime
<<time>> + 1-T: IntervalTime
<<time>> + 0-T : PointTime
......
DataBaseTime
<<time>> + 1-T: IntervalTime
<<time>> + 0-T : PointTime
......
Point
<<position>> + x : Real
<<position>> + y : Real
.....
SpatioTemporal
Attribute:
Operation():
SpatioTemporalAttribute
Attribute:
Operation():
2.. *
{sequence}
1
0, 1
0 .. *
ZeroTCell
<<position>> + pointid: Point
<<time>> # systemTime: 1-T
<<dynamic>> + create(P: Point): Boolean
<<dynamic>> + kill(P: ZeroTCell): Boolean
<<dynamic>> + destroy(P: ZeroTCell)
<<dynamic>> + reincarnate(P: ZeroTCell): Boolen
......
2
0 .. *
OneTCell
<<index>> + pointSequence: sequence(Point)
<<time>> # systemTime: 1-T
<<query>> + /length : Real
<<index>> + parent: OneTCell
<<dynamic>> + create(P: Sequence(Point)): Boolean
<<dynamic>> + kill(P: OneTCell): Boolean
<<dynamic>> + destroy(P: OneTCell): Boolean
<<dynamic>> + reincarnate(P: OneTCell): Boolean
....
{set}
1 .. *
0,2
1
Parent
1 ..*
Child
1
Parent
1..*
Child
TwoTCell
<<index>> + onetcell: set(OneTCell)
<<time>> + systemtime: 1-T
<<query>> + /area : Real
<<query>> + /perimeter : Real
<<index>> + parent: TwoTCell
<<dynamic>> + create(P: Set(OneTCell)): Boolean
<<dynamic>> + kill(P: TwoTCell): Boolean
<<dynamic>> + destroy(P: TwoTCell): Boolean
<<dynamic>> + reincarnate(P: TwoTCell): Boolean
....
1
0,1
0,1
TemporalCellTuple
<<topology>> + zerotcellid : ZeroTCell
<<topology>> + oneotcellid : OneTCell
<<topology>> + twotcellid : TwoTCell
<<time>> + systemtime : 1-T
....

Figure 2: Conceptual schema for cell tuple based spatio-temporal data model [adpated from (Raza and Kainz, 1999)].

Three dimensions of time are incorporated as three specialised classes of LinearTimeClass, i.e., DataBaseTimeClass (DT), WorldTimeClass (WT) and SystemTimeClass (ST). ST reflects the time at which spatial changes occur in the system. ST is explicitly associated with the spatial object and is independent of DT and WT. It is different from DT in the sense that the latter represents the updating of STAO in the database, while the former indicates the updating of the spatial object. In a LinearTimeClass two data types are defined, PointTime [0-T] and IntervalTime [1-T]. The object of SpatioTemporalClass can be defined as:

An (open) *n*-tcell is a topological space homeomorphic to an open ball E^n of $\mathbb{R}^n$. A ZTC is a temporal node, a OTC is temporal arc, a TTC is a temporal polygon/area. ZTC, OTC and TTC are members of the temporal cell complex (TCC). A finite collection *k* of *n*-tcells is a TemporalCellComplex if

- Different elements of k have disjoined interiors.
- For each *n*-tcell in *k* the boundary of *n* is a union of elements of *k*.
- If $n, m \in k$, and $n \cap m = \neg\emptyset$, then $n \cap m$ is a tcell, and is a union of elements of *k*. Where tcell is either an *n*, *m* (of different dimension) or a common face.

The objects ZTC, OTC and TTC define their own spatial and temporal configurations.

- ZTC: A zero-dimensional object, which has a position and is represented by one PointObject. ZTC's life span is represented by interval time 1-T [T_{From},T_{Until}].
- OTC: A one-dimensional object, which is bounded by two 0-tcells. The OTC object is an ordered sequence of point objects. In case of two point objects the last and first points are the first and last 0-tcell. In case of a loop, the first and last point or ZTC is the same. In the parametric form a OTC is an ordered sequence of points,
  $\{x \mid x = p_1, p_2, p_3, \ldots, p_i, \ldots\ldots p_n\}$
  where $p_i \neq p_{i-1}$ and $0 < z < 1$ and $x = p_1 + (p_2 - p_1) * z$, $x = p_2 + (p_3 - p_2) * z$, ……, $x = p_{n-1} + (p_n - p_{n-1}) * z$
  Points other than the first and last point are called intermediate points, which form the shape of OTC. A loop must have two intermediate points. There is no limit (depending upon implementation) of intermediate points for OTC.
- TTC: A two dimensional object bounded by a closed cycle of ZTCs and OTCs. The life of each TTC is depicted by 1-T [T_{From}, T_{Until}].

All objects ZTC, OTC and TTC can either be born or die. Consistency rules governing these objects can be found in Raza and Kainz (1999). T_{Until} could be NULL which shows the object is still alive (active). If T_{Until} is not-null, then it indicates that the object is dead (inactive). Therefore, all active objects are represented with timestamp [T_{From}, *] and inactive with [T_{From},T_{Until}]. An active *n*-tcell is not allowed to have an inactive boundary *(n-1)*-tcell. Whereas, an inactive *n*-tcell can have an active boundary of *(n-1)*-tcell. For example, an active OTC can not

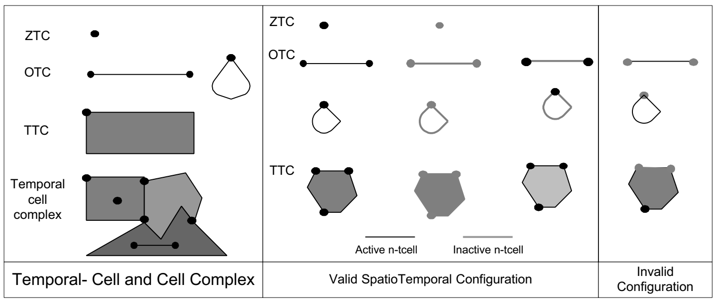


Figure 3. Valid and invalid spatio-temporal configuration.

have inactive ZTC(s), while, an inactive OTC can be defined by active ZTC(s). Figure 3 is the illustration of some simple valid and invalid spatio-temporal configuration for ZTC, OTC, TTC. Active (alive) objects are indicated by black shade and dead (inactive) by gray shade. This imposes consistency constraints while constructing or updating the spatio-temporal database. As the object is defined as a spatio-temporal object, the topological relations could be defined as spatio-temporal topological relations, i.e., the spatial relations which are valid over time. The TemporalCellTupleClass preserves the temporal cell tuple structure (Figure 2). A temporal cell tuple structure encapsulates the spatio-temporal topology of each spatio-temporal object. A temporal cell tuple (TCT) is a set of C and T {C, T}, where C is a set of cells $\{c_0, c_1, c_2, \ldots c_n \mid c_i \in \text{TCC}\}$ and T is a time interval $\{T_{From}, T_{Until} \mid (T_{From} < T_{Until}) \wedge (T_{From}, T_{Until} \in \text{ST})\}$ or 1-T. Therefore, TCT = $\{c_0, c_1, c_2, \ldots c_n, T_{From}, T_{Until}\}$.

3 OPERATIONS FOR SPATIOTEMPORALCLASS

Two types of operators can be defined, i.e., static and dynamic operators. A static operator does not effect the system's state or status of spatio-temporal objects, e.g., query operators (calculating the length, area, time period, boundary or co-boundary, etc). On the other hand, dynamic operators change the state of the system or status of the spatio-temporal objects, e.g., creating, deleting or updating an *n*-tcell. Normally, in atemporal GIS, three fundamental dynamic operations are performed, i.e., Insert, Delete and Update. All dynamic operators are the variation of one of these operations (Worboys, 1997). Unlike atemporal GIS, in a TGIS an object may die or be killed, but they remain in the database with a certain time stamp indicating their life span. As mentioned earlier any *n*-tcell object can be born or die. Therefore, four fundamental dynamic operators can be distinguished in spatio-temporal databases, i.e., **Create**, **Kill, Reincarnate** or **Delete (Destroy)** the objects (ZTC, OTC or TTC) of respective class. In spatio-temporal databases the kill operation is different from delete operation, as the latter is merely a purge operation. Updating of spatio-temporal objects is complex, any update operation effects the other objects, particularly in unified approach. Any spatial change is the result of creation (birth) and / or destruction (death) of a *n*-tcell. **Create** operator is equivalent to the usual insert operators. The task of this operator is to create a new and/or update an exiting object. This operator specifies time stamp [start, *] of each spatial object, where upper bound of time interval is undefined (*). All objects with [start, *] time stamps are called active objects. The **Kill** operator, kills the spatio-temporal objects by defining the upper bound of the time interval. Objects after being killed are called inactive objects. These objects remain in the database only for the query purpose or Reincarnate operator. Therefore, upper bound with (*) of ST is replaced by current system time. The operator **Destroy** or Delete, permanently deletes the spatio-temporal object from the database. Therefore, they are no longer available for any type of operations (static or dynamic). The operator **Reincarnate** turns an inactive object to an active object by replacing the upper bound of time interval to (*). The Create and Kill operators are discussed in this paper. Generally in a spatial data model the Create (update) operation is performed by checking the intersection of spatial objects (node, arc or polygon). Spatio-temporal data models demand different treatment of spatial objects, because the existing ones are not thrown out, they are persevered with valid time stamp. This Create operation is basically an overlay operation. The overlay operation in spatio-temporal databases is much more complex than its spatial counterpart. Computationally, polygon-polygon operations is most challenging task in vector type spatial database (Laurini and Thompson, 1992).

T1 \ T2	ZTC	OTC	TTC
ZTC	X	√	√
OTC	√	√	√
TTC	√	√	√

Figure 4. Possible intersection for *n*-tcell.

In the unified spatio-temporal data model, when a ZTC, OTC or TTC is inserted, the following scenario can be expected. A ZTC may intersect with ZTC, OTC or TTC, a OTC may intersect with ZTC, OTC or TTC and a TTC may intersect with ZTC, OTC or TTC. Figure 4 shows all nine possibilities when a *n*-tcell at time T1 may intersect with a *n*-tcell at time T2. The cross (X) shows invalid intersection or overlay, i.e., geometrically ZTC can not intersect with ZTC object (node can not intersect a node). The tick (√) is a valid overlay operation, e.g., a ZTC can intersect with a OTC or TTC object. In each case there are various possibilities, e.g., ZTC may intersect at the boundary of OTC or the interior of OTC. The three topological invariant of spatial objects (*n*-tcells) are boundary, interior and exterior. This point set topology approach is employed to analyze these intersections. Only boundary (∂) and interior (°) of OTC and TTC are considered to investigate these intersections. The intersection at the exterior of any *n*-tcell is straightforward. The boundary of ZTC is empty. Create and Kill operators needed for ZeroTCell- and OneTCell-Class for 2-TemporalCellComplex are discussed in this paper. These operations are closed as intersections of two *n*-tcells ($0 \leq n \leq 2$) always produces an *n*-TemporalCellComplex. It is assumed that all *n*-tcells are inside the universal (void) 2-tcell (∅). The syntax for each operation of the respective class is given in Figure 2.

4 ZEROTCELLCLASS

4.1 ZTC insertion (create operation)

Various possible scenarios and actions while inserting a ZTC are discussed in sequel.

4.1.1 ZTC coincidence: This case is trivial because the ZTC can not be added to an existing one.

4.1.2 ZTC with OTC: There are four possibilities (Figure 5).
a] At a boundary of OTC: Operation is rejected, as it is similar to ZTC to ZTC insertion. b] At intermediate point of OTC: Kill OTC (*a1*) and TCT (c1 and c2). Create ZTC (*n3*), two OTC (*a2* and *a3*) and TCT (c3, c4, c5 and c6).
c] At point between the intermediate points of OTC: Create new point object (p'), new ZTC (*n3*) and new TCT (c3, c4, c5 and c6).
d] At point between of two ZTC: Same as # 3.

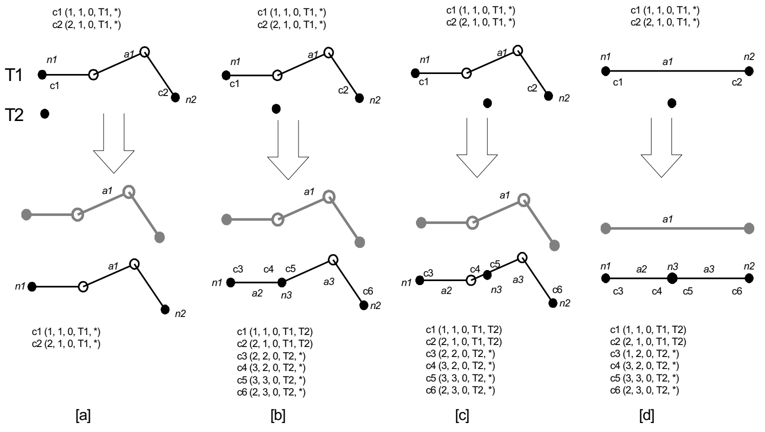


Figure 5. Create ZTC: ZTC intersect with OTC.

4.1.3 ZTC with TTC: A ZTC may intersect with TTC at the boundary or interior of TTC.
1] At boundary of TTC: The boundary of TTC is OTC, therefore it is similar to ZTC to OTC intersection, where we had three possibilities. ZTC can intersect at:
a] The boundary of OTC: Operation is rejected, as it is similar to ZTC to ZTC insertion.
b] The point between the intermediate points or between ZTC of OTC (Figure 6): Kill TTC (A), OTC (*a2*), TCT (c1, c3, c5,…, c8). Create a point object (p'), a ZTC (*n3*), a OTC (*a3, a4*), a TTC (A') and TCT (c9, c10, .. c18).
c] The intermediate point of OTC: Same as [b], expect no point object is created.

2] The interior of TTC: The process is shown in Figure 7. Only new ZTC and associated TCT is added.

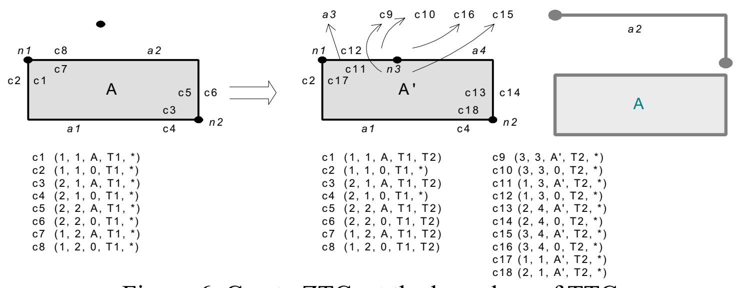


Figure 6. Create ZTC: at the boundary of TTC.

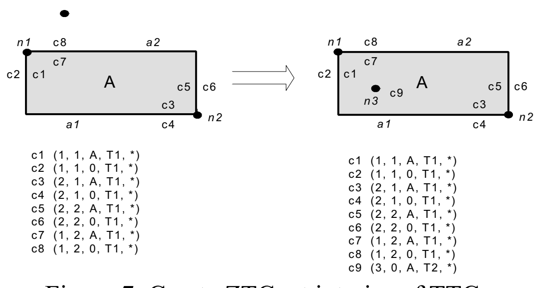


Figure 7. Create ZTC: at interior of TTC.

4.2 ZTC Kill Operator (f1)

A ZTC (isolated) can be killed when it is not a boundary or face of OTC by simply closing the upper bound of the time interval. The case of ZTC being a face of OTC is discussed as follows.
1] ZTC associated with OTC: A ZTC can be associated with OTC in following ways (Figure 8).
A ZTC can be a boundary of single OTC (Figure 8[a]), a ZTC can be a boundary of two OTCs (Figure 8[b]) or a ZTC can be a boundary of more than two OTCs (Figure 8[c]). In all cases operation is discarded, as an active OTC can not have an inactive ZTC.

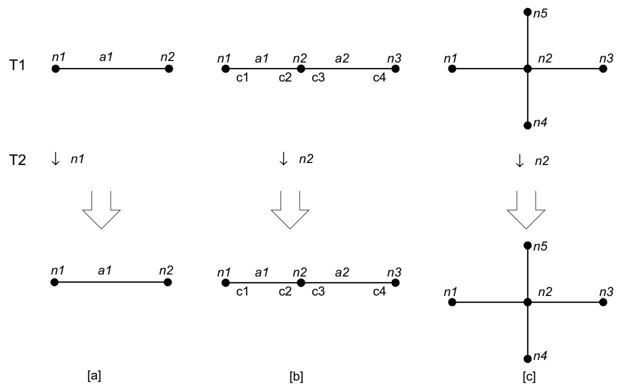


Figure 8. Kill ZTC: ZTC is a boundary of OTC.

2] ZTC associated with TTC: Same convention applies as in case of ZTC associated with OTC. Therefore, no further elaboration is considered.

5 ONETCELLCLASS

5.1 OTC Insertion (create operator)

Insertion of OTC is a recursive operation, which starts by insertion of boundary of OTC, i.e., zero, one or two ZTC. Like ZTC, this operation can also be viewed from three perspectives, i.e., when OTC intersects with ZTC, OTC and TTC. The intersection of ZTC with OTC and OTC with ZTC is not symmetric, like in spatial data structure. In spatio-temporal case it is vital that what exist first, i.e., ZTC or OTC.

5.1.1 OTC with ZTC:

There are three possibilities:

a] ZTC intersect at boundary of OTC (Figure 9[a]): If one of the boundaries of OTC coincides with ZTC (*n1*), then the *n1* become a boundary of new OTC (*a1*). New ZTC (*n2*) is added which becomes a second boundary of OTC (*a1*). The TCT c1 is killed and new TCT (c2 and c3) are added.

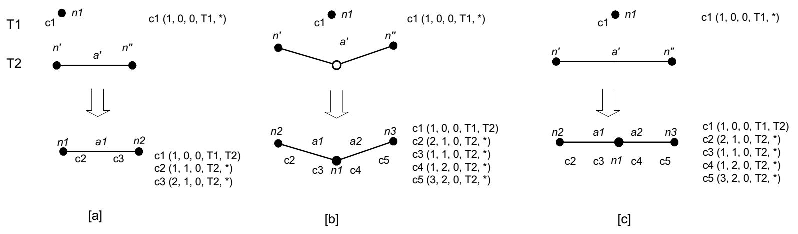


Figure 9. Create OTC: OTC intersect with ZTC.

b] ZTC intersect at intermediate point of OTC (Figure 9[b]): Two new ZTC (*n2* and *n3*) are added. Two new OTC are added, i.e., *a1* and *a2*, with boundaries <*n2,n1*> and <*n1,n3*>, respectively. TCT c1 is killed. New TCTs (c2, c3, c4 and c4) are added.

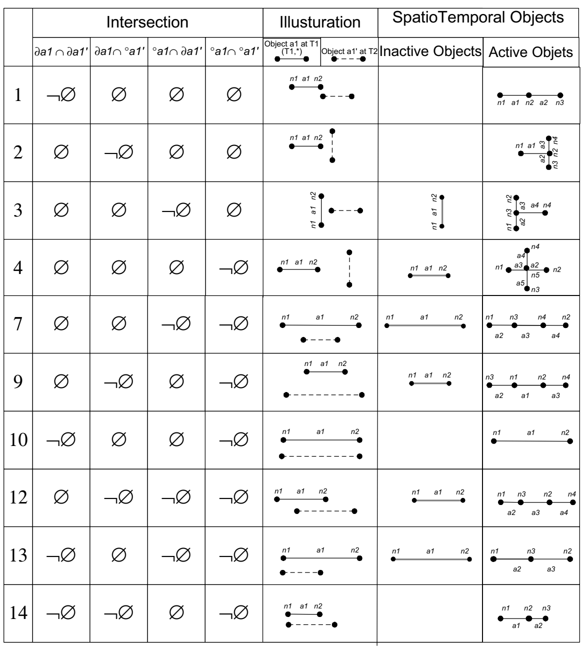
c] ZTC intersect at point between intermediate or ZTC of OTC (Figure 9[c]): Two new ZTC (*n2* and *n3*) are added. Two new OTCs (*a1* and *a2*) are added, with boundaries <*n2,n1*> and <*n1,n3*>, respectively. TCT c1 is killed. New TCTs (c2, c3, c4 and c4) are added.

5.1.2 OTC intersect with OTC:

This operation could be very complex, i.e., there are number of possibilities when OTC at T1 can intersect with OTC at T2. To simplify, we use a point set approach. Let *a1* be a OTC (with no intermediate points) with ∂*a1* and °*a1* at time T1 and *a1*' be a OTC with ∂*a1*' and °*a1*' at time T2. A 2x2-intersection can be defined as.

$$\mathrm{I4} = \begin{pmatrix} \partial a1 \cap \partial a1' & \partial a1 \cap {}^{\circ}a1' \\ {}^{\circ}a1 \cap \partial a1' & {}^{\circ}a1 \cap {}^{\circ}a1' \end{pmatrix}$$

When we consider the empty [∅] and non-empty [¬∅] intersection (single) of two objects. There are $2^4 = 16$ possible intersections between OTC at time T1 and OTC at time T2. Not all are applicable or valid. Some of them are symmetric. The valid cases are when: 1] ∂*a1* intersects with ∂*a1*', 2] ∂*a1* intersects with °*a1*', 3] °*a1* intersects with ∂*a1*, 4] °*a1* intersects with °*a1*', 7] °*a1* intersects with ∂°*a1*', 9] °*a1*' intersects with ∂°*a1*, 10] ∂*a1* intersects with ∂*a1*' and °*a1* intersect with °*a1*', 12] °*a1* intersects with ∂°*a1*' and ∂*a1* intersects with °*a1*', 13] °*a1* intersects with °*a1*' and ∂*a1*' intersects with ∂°*a1*, and 14] ∂*a1* intersects with ∂°*a1*' and ∂*a1*' intersects with °*a1*. Without going into details of all the intersections, only valid intersections are considered and case 1,2,3 and 4 are discussed in details. However, OTC is extended to *n*-segments. To simplify, the orientation of OTC at T1 and T2 is assumed to be the same.



	Intersection				Illusturation		SpatioTemporal Objects	
	∂a1 ∩ ∂a1'	∂a1∩ °a1'	°a1∩ ∂a1'	°a1∩ °a1'	Object a1 at T1 (T1,*)	Object a1' at T2	Inactive Objects	Active Objets
1	¬∅	∅	∅	∅				
2	∅	¬∅	∅	∅				
3	∅	∅	¬∅	∅				
4	∅	∅	∅	¬∅				
7	∅	∅	¬∅	¬∅				
9	∅	¬∅	∅	¬∅				
10	¬∅	∅	∅	¬∅				
12	∅	¬∅	¬∅	¬∅				
13	¬∅	∅	¬∅	¬∅				
14	¬∅	¬∅	∅	¬∅				

Figure 10. OTC-OTC intersection (single segment).

1] Boundary of OTC intersects with boundary of OTC: A ZTC (*n3*), a OTC (*a2*) and TCTs (c3 and c4) are added (Figure 11).

2] Boundary of OTC intersects with interior of OTC: Two new ZTC (*n3* and *n4*) and a new OTC (*a2* and *a3*) are added by defining boundaries as <*n3, n2*> and <*n2, n4*>, respectively. Exiting ZTC (*n2*) is used in defining the boundaries of OTC. New TCT (c3, c4, c5 and c6) are added (Figure 12).

3] Interior of OTC intersects with boundary of OTC: The boundary of OTC (*a1'* at time T2) can intersect with interior of OTC (*a1* at time T1) in two ways.

a] When it intersects at the intermediate point, a new ZTC is generated and existing OTC (*a1*) divided into two parts (*a2* and *a3*), while *a1'* remains unchanged (Figure 13[a]). OTC *a1* and TCT c1 and c2 are killed. ZTC (*n3*), three new OTC (a2, a3 and a4) are added and new TCT (c3,..c6) are added.

b] When *a1'* intersect at any point between the intermediate point or ZTC of *a1* (Figure 13[b]). A new point and ZTC is created and *a1* is divided into two parts (*a2* and *a3*). OTC *a1*, TCT c1 and c2 are killed. New point, ZTC (*n3*), three new OTC (*a2, a3* and *a4*) and new TCT (c3, .., c6) are added .

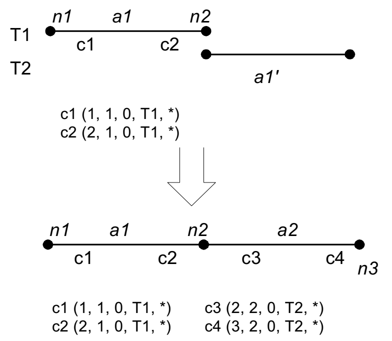


Figure 11. Create OTC: Boundary – boundary intersects of OTC.

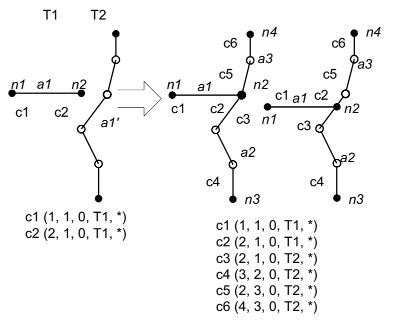


Figure 12. Create OTC: Boundary-interior intersects of OTC.

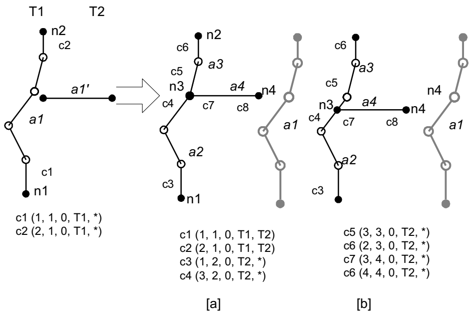


Figure 13: Create OTC: Interior-boundary intersects of OTC.

4] Interior of OTC intersects with interior of OTC: Interior of OTC (*a1*) at time T1 can intersect with interior of OTC (*a'*) at time T2 in following ways:
a] Both intersect at intermediate point (Figure 14[a]).
b] Intermediate point of *a'* intersects at point between intermediate point of *a1* (Figure 14[b]).
c] Intermediate point of *a1* intersects at point between intermediate point of *a'* (Figure 14[c]).
d] Intersect at the intermediate point of both *a1* and *a'*(Figure 14[d]).

First three cases (a, b and c) have the same steps for insertion of OTC except the third case where a new point calculated as intersection. All results in four new OTCs. Kill OTC (*a1*), TCT (c1 and c2). Add ZTC (*n3, n4* and *n5*), OTC a2, a3, a4 and a5 with boundaries <*n1,n4*>, <*n4, n2*>, <*n3,n4*> and <*n4,n5*>, respectively and TCT (c3, c4, …, c10). In the forth-case (d) five new OTCs are generated as shown in Figure 14[d].

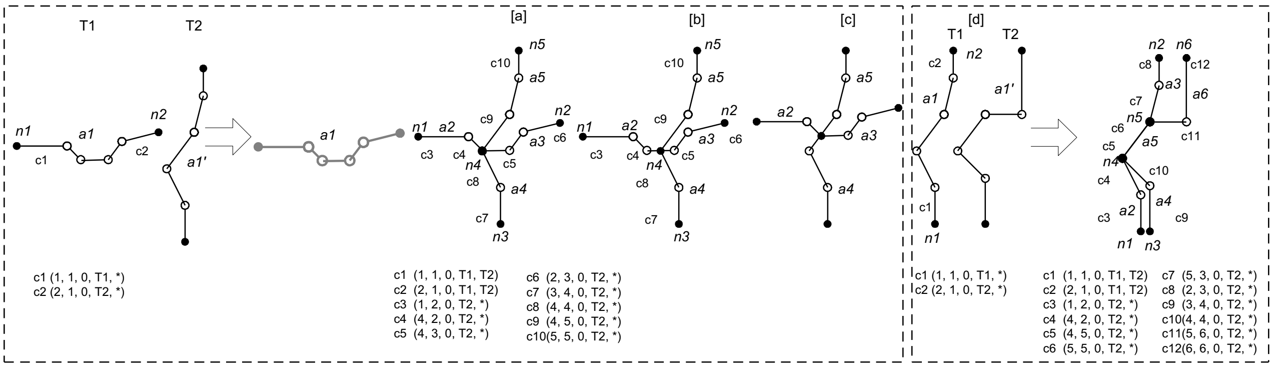


Figure 14. Create OTC: Interior-interior intersects of OTC.

5.1.3 OTC with TTC: Same conventions are employed as in the case of OTC-OTC intersection. OTC (*a*) at time T1 can intersect with TTC (A) at time T2 in many ways. This can be expressed by a 2x2 intersection metrix.

$$\mathrm{I4} = \begin{pmatrix} \partial a \cap \partial \mathrm{A} & \partial a \cap {}^{\circ}\mathrm{A} \\ {}^{\circ}a \cap \partial \mathrm{A} & {}^{\circ}a \cap {}^{\circ}\mathrm{A} \end{pmatrix}$$

Like OTC, this, too, provides 16 possible intersections, but all are not valid. Five valid intersections exist when 1] ∂a intersects with ∂A, 2] ∂a intersects with °A, 3] °*a* intersects with ∂A, 4] ∂a intersects with °A and °*a* intersects with ∂A, 5] °*a* intersects with ∂A and °*a* intersects with °A. Details of case-1 is provided here. Only single boundary of OTC and TTC is considered here.

1] Single boundary of OTC intersect with of TTC: Single boundary of OTC can intersect with the boundary of TTC in two ways, i.e., a] intersects at

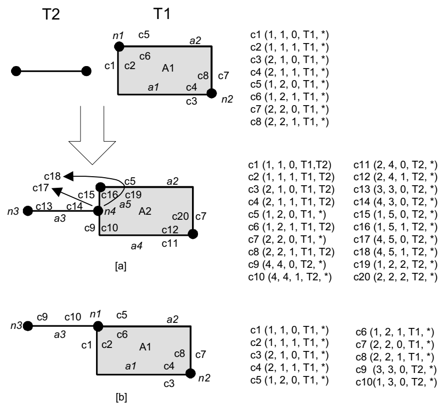


Figure 15. Create OTC: Boundary of OTC intersects with boundary of TTC.

boundary of TTC Figure 15[a], b] intersects at boundary of OTC of TTC Figure 15[b]. In the first case, the creation of OTC yields a new TTC, while in the latter no new TTC is created.

a] Intersect at boundary of TTC: Kill TTC (A1), OTC (*a1*), TCT (c1, c2, c3, c4, c6 and c8). Create ZTC *(n3* and *n4*), OTC (*a3, a4* and *a5*), TTC (A2) TCT (c7, c9, c10, .., c20).
b] Intersect at boundary (ZTC) of OTC: Create ZTC (*n3*), OTC (*a3*) and TCT (c9 and c10).

5.2 OTC Kill Operator (fl)

Killing an isolated OTC is straightforward, as the null (*) value of the upper bound of time interval (system time) is replaced by current time. Cases when the boundary (ZTC) of OTC belongs to another OTC can be complex. An OTC itself can be a boundary of TTC. These two cases are discussed below.

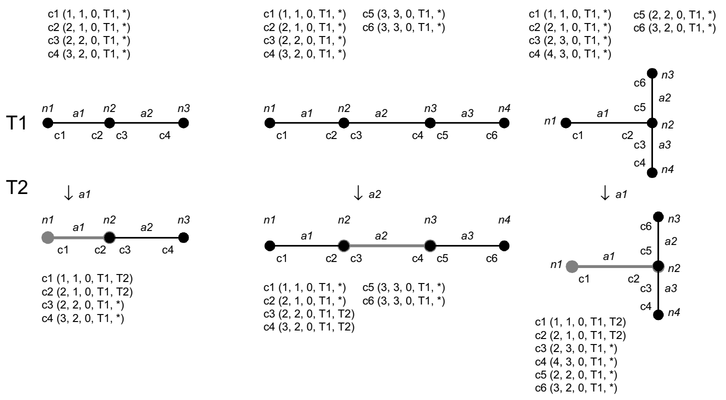


Figure 16. Kill OTC: Boundary of OTC shared by another OTC.

5.2.1 Boundary of OTC shared by another OTC: a] Shared by one OTC (Figure 16[a]): Kill ZTC (*n1*), OTC (*a1*) and TCT c1 and c2). b] Shared by two OTCs (Figure 16[b]): Kill OTC (*a2*) and TCT (c3 and c4). c] Shared by two or more OTC (Figure 16[c]): Kill ZTC (*n1*), OTC (*a1*) and TCT (c1 and c2).

5.2.2 OTC is a boundary of TTC: a] TTC is defined by a single OTC. b] TTC is defined by more than one OTC. c] OTC is shared by two TTC. In all three cases (Figure 17) the Kill operation is discarded (enforce spatio-temporal consistency rule), as the OTC is the boundary of TTC and TTC can not have an inactive boundary).

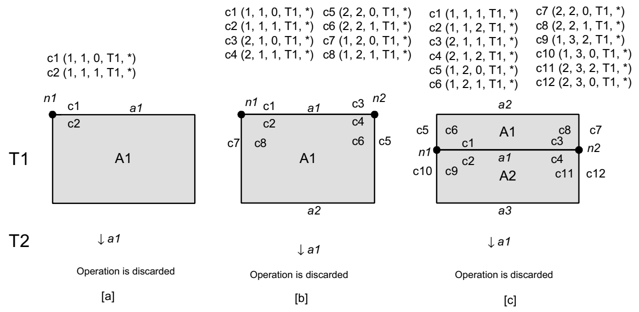


Figure 17. Kill OTC: OTC is a boundary of TTC.

6 CONCLUSION

We present a novel approach based on state-of-the-art OO concepts, UML and the mathematical theory of cell complexes. Four dynamic operators, i.e., create, kill, destroy, and reincarnate have been identified to update a spatio-temporal data model. These operators are designed for a unified cell tuple based spatio-temporal data model. Singularities of *n*-tcells are incorporated. To discern the various geometric intersections between two *n*-tcells, the notion of point set approach is employed. Various scenarios are discussed while applying create and kill operators. The results of any create or kill operator produces a *n*-TemporalCellComplex and guarantees the spatio-temporal consistency in the database. One of the chief advantages of this approach is that it may provide a basis for implementing any generic TGIS.

7 REFERENCES

Booch G., 1994, Object-Oriented Analysis and Design with Application, The Benjaman/Cummings Publishing Company, Inc.

Laurini R., and Thompson, D., 1992, Fundamentals of Spatial Information Systems, Academic Press Limited, London.

Raza A., and Kainz W., 1999, Cell Tuple Based Spatio-Temporal Data Model: An Object Oriented Approach, Eight ACM Conference on Information and Knowledge Management (CIKM'99) and Symposium on Geographic Information System (GIS'99), November 2-6th, Kansas City, USA.

UML 1.3, 1999, http://www.rational.com/uml/index.jtmpl (10 March 1999).

Worboys M., 1997, GIS: A Computing Perspective, Taylor & Francis.