Maintaining Consistent Topology including Historical Data in a Large Spatial Database

Peter van Oosterom
Cadastre Netherlands, Company Staff
P.O. Box 9046, 7300 GH Apeldoorn, The Netherlands.
Phone: +31 55 528 5163, Fax: +31 55 355 7931.
Email: oosterom@kadaster.nl

This paper describes a data model and the associated processes designed to maintain a consistent database with respect to both topological references and changes over time. The novel contributions of this paper are: 1. use of object identifiers composed of two parts: oid and time; 2. long transactions based on a check-out/check-in mechanism; and 3. standard SQL (structured query language) enhanced with SOL (spatial object library) for both the batch production of update files and for the interactive visualization of the changes over time.

1 Introduction

Large scale Topographic and Cadastral data in the Netherlands [9] are stored and maintained in *one integrated system* based on the relational database CA-OpenIngres with the spatial object library (SOL) [4] and X-Fingis [10, 11, 13]. Storing and maintaining consistent topological relationships is important in a spatial database. Topology is essential to the nature of the Cadastre: parcels may not overlap and parcels should cover the whole territory. About 400 persons (surveyors, cartographers) are updating these data simultaneously. After the initial delivery of all data, the customers get periodic updates of the database. Without storing object-history in the database, these *update files* are difficult to extract [16]. Historical data is also used to find the previous owners of a certain polluted spot. This illustrates the need for consistently maintaining both time and topology in the database.

General introductions to spatio-temporal modeling are given in [14, 18, 21][1]. Although several authors have described a spatial-temporal data model and query language, they ignore the problem of maintaining the data in their models, which is complicated due to the topology references. Our data model based on topology and history is presented in Section 2. Topological editing of information is discussed in Section 3, in which particular attention is paid to the fact that multiple users must be able to work simulta-

[1] A glossary of temporal terms in databases can be found in [8].

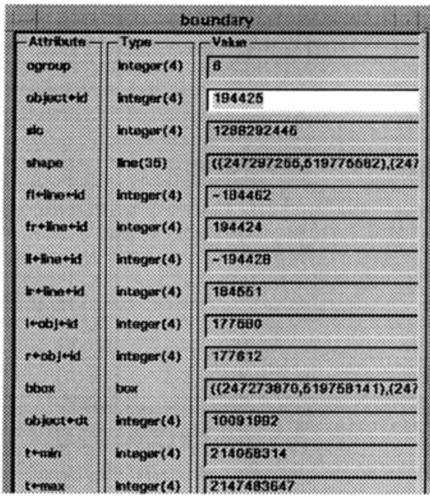

Fig. 1: Boundary record 194425

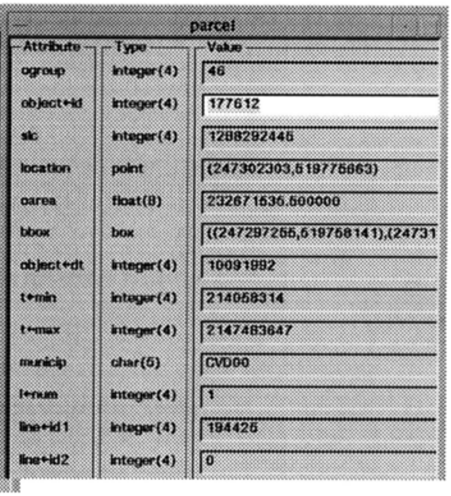

Fig. 2: Parcel record 177612

neously. The production of update files using standard SQL (structured query language) is described in Section 4. In contrast to these 'batch' type of jobs, some possibilities for interactive visualizations of changes over time are given in Section 5 together with other future work. Finally, conclusions can be found in Section 6.

2 Data model

Integrated storage of all components of the data (metric information, topology, thematic attributes, and historic information) in one database is the key property, which enables controlling data consistency. Example records are shown in Fig.1 and 2: `boundary` with parcel boundaries and `parcel` with additional parcel information. Note the integrated use of traditional data types and spatial data types, such as `point`, `line`, and `box` in the data model. In the data model all objects get a unique identifier `object_id`[2], which enable efficient communication with customers of the update files.

Topological references

In theory, explicitly storing planar topological information (references) causes data redundancy, because the references can be derived from accurate metric information as stored in the `shape` attribute of type `line(35)` in the `boundary` table and in the `location` attribute of type `point` in the `parcel` table. However, explicitly storing the topological references makes checking the topological structure (data quality) feasible within the database. Further, it is also convenient for data manipulation; e.g. compute the polygon [3] or find neighbors of a face.

[2]The `object_id` is unique within each group of an object type `ogroup` and is maintained nation-wide. Sometimes in this paper the pair `ogroup, object_id` is abbreviated to just `oid` for simplicity.

[3]The terms *face*, *edge*, and *node* are

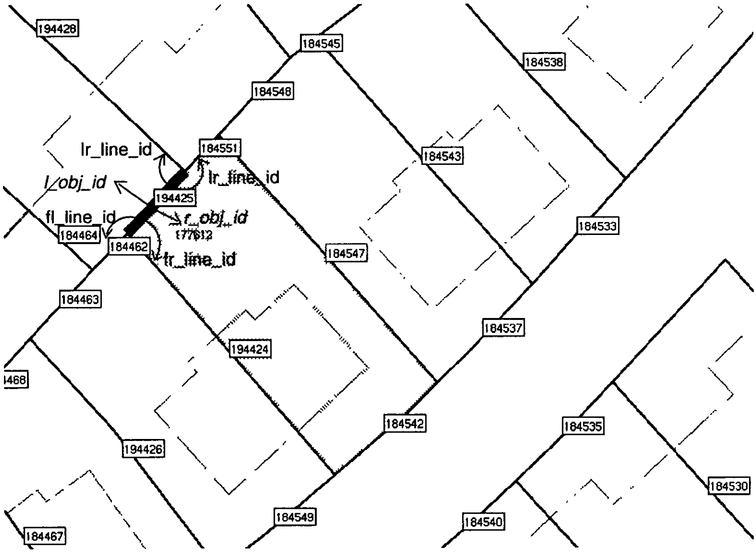

Fig. 3: GEO++ screendump with example boundary record

The spatial basis of the data model is a planar topological structure, called the *CHAIN-method* [15], similar to the *winged edge structure* [3]; see Figs. 1, 2, and 3. However, all references to edges are *signed* (+/−), indicating the direction of traversal when forming complete boundary chains. The edges contain four references to other edges: in the `boundary` table there are attributes to indicate the immediate left and right edge at the first point (`fl_line_id` and `fr_line_id`) and the immediate left and right edge at the last point (`ll_line_id` and `lr_line_id`). Further, references from a face to the first edge of its boundary chain and, if islands are present, references to the first edge of every island-chain are stored. In this model polygons related to faces can be composed by using the signed references only. So, without using geometric computations on the coordinates. Besides the references from faces to edges, and from edges to edges, there are also references from edges to left and right faces: `l_obj_id` and `r_obj_id` in the `boundary` table. A bounding box `bbox` attribute is added to every table with spatial data in order to implement efficient spatial selection. Finally, the computed area is stored in the `oarea` attribute of the `parcel` table.

Historical information

The updates in our database are related to changes of a discrete type in contrast to more continuous changes such as natural phenomena or stock rates. The number of changes per year related to

used when the topological aspects are intended. The terms *polygon*, *polyline*, and *point* are used when discussing the metric aspects. Finally, terms such as *parcel* and *boundary* are used to refer to the objects.

the total number of objects is about 10%. It was therefore decided to implement history on tuple level[4]. This in contrast to implementing history on attribute level, which requires specific database support or will complicate the data model significantly in a standard relational database; see [19, 14, 20, 27]. In our model every object is extended with two additional attributes: `tmin` and `tmax`[5]. The object description is valid starting from and including `tmin` and remains valid until and excluding `tmax`. Current object descriptions get a special value `MAX_TIME`, indicating that they are valid now. `MAX_TIME` is larger than any other time value. There is a difference between the *system (transaction)* time, when recorded object changed in the database, and the *valid (user)* time, when the observed object changed in reality. In the data model `tmin/tmax` are system times. Further, the model includes the user time attribute `object_dt` (or `valid_tmin`) when the object was observed. Perhaps in the future also the attributes `last_verification_dt` and `valid_tmax` could be included, which would make it a *bitemporal* model.

When a new object is inserted, the current time is set as value for `tmin`, and `tmax` gets a special value: `MAX_TIME`. When an attribute of an existing object changes, this attribute is not updated, but the complete record, including the `oid`, is copied with the new attribute value. Current time is set as `tmax` in the old record and as `tmin` in the new record. This is necessary to be able to reconstruct the correct situation at any given point in history. The *unique identifier* (key) is the pair `(oid, tmax)` for every object version in space and time.

For the topological references, only the `oid` is used to refer to another object and not `tmax`. In the situation that a referred object is updated and keeps its `oid`, then the reference (and therefore the current object) does not change. This avoids, in a topologically structured data set, the propagation of one changed object to all other objects as all objects are somehow connected to each other. In case the `oid` of a referred object has changed (becomes a different object), the referring object is also updated and a new version of the referring object is created.

The following example shows the contents of a database, which contained on `12 jan` one line with `oid` 1023. On `20 feb` this line was split into two parts: 1023 and 1268; see Fig. 4. Finally, the attribute `quality` of one of the lines was changed on `14 apr`. The SQL-queries in Section 4 show how easy it is to produce the update files with new, changed, and deleted objects related to a specific time interval.

[4]Instead of storing the old and new states, it is also possible to store the events only [7, 1]. However, it will not be easy to retrieve the situation at any given point in time.

[5]This is similar to the Postgres model [23]. A temporal SQL extension is described in [22]. In [26] a temporal object database query language for spatial data is presented.

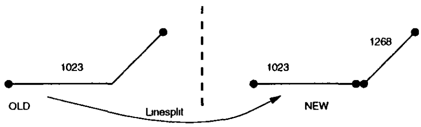

Fig. 4: A 'line' split into 2 parts

line

oid	shape	... quality	tmin	tmax
1023	(0,0),(4,0),(6,2)	1	12jan	20feb
1023	(0,0),(4,0)	1	20feb	14apr
1268	(4,0),(6,2)	1	20feb	MAX_T
1023	(0,0),(4,0)	2	14apr	MAX_T

Predecessor and successor

A query producing all historic versions of a given object only needs to specify the `oid` and leave out the time attributes. This does work for simple object changes, but does not work for splits, joins, or more complicated spatial editing. However, this information can always be obtained by using spatial overlap queries with respect to the given object over time, that is, not specifying `tmin/tmax` restrictions.

3 Locking, check-out, and check-in

A GIS is different from many other database applications, because the topological edit operations can be complicated and related to many old and new objects. This results in *long transactions.* During this period other users are not allowed to edit the same theme within this rectangular work area. They must also be allowed to view the last correct state before the editing of the whole database. An alternative to locking is versioning [5], but it is impossible to merge conflicting versions without user intervention. Therefore, the edit locking strategy is used and this is implemented by the table `lock`.

As the database must always be in a consistent state, it may not be polluted with 'temporary' changes that are required during the topological edit operations. This is the motivation for the introduction of a *temporary work copy* for the GIS-edit program; e.g. X-Fingis [10, 11, 13]. The copy is made during *check-out* and is registered in the `lock` table. This is only possible in case no other work areas overlap the requested region with respect to the themes to be edited. The database is brought from one (topologically) consistent state to another consistent state during a *check-in.* It is important that all changes within the same check-in get the same time stamps in `tmin/tmax` (system time as always). This architecture also has the advantage that it enables an easy implementation of a high level 'cancel' operation (rollback).

Locking a work area

What exactly should be locked when a user specifies a rectangular work area? Of course, everything that is completely inside the rectangle must be locked. This is achieved at the *application* level: check-out and check-in. Objects that cross work area boundaries could also be locked, but this may affect a large part of the database. Other users may be surprised to see when they want to check-out a new non-overlapping part (rectangle), this is impossible due to elongated objects that are locked. Therefore, the concept of *partial locks* is introduced for these objects: the *coordinates* of the line segment crossing the boundary of the work area are not allowed to change. Together with the fact that the rectangular work areas can never overlap, this implies that the other changes to the edges and faces that cross the borders of two work areas are *additional* and can be merged in the database. Therefore these objects do not have to be locked, but have to be checked in with some additional care. It is possible that two check-ins want to modify the same object; see Fig. 5. If no care is taken and both check-ins replace the object, then only the second version is stored and the changes from the first are lost. Therefore, the following steps must be taken for every changed object crossing the work area boundary:

- refetch the object from the database and acquire a *database* update lock for this object;
- if other changes have occurred, then 'merge' these with the work area version of objects;
- reinsert the 'merged' object in database and release the database update lock.

The 'solution' for avoiding deadlocks, is to allow only one check-in at a time (check-in queue). So, all check-ins are processed sequentially.

Errors and improvements
Errors in the past with respect to data collecting or entering pose a difficult problem: should these be corrected by changing the history `tmin/tmax`? Because of possible consistency problems it was decided

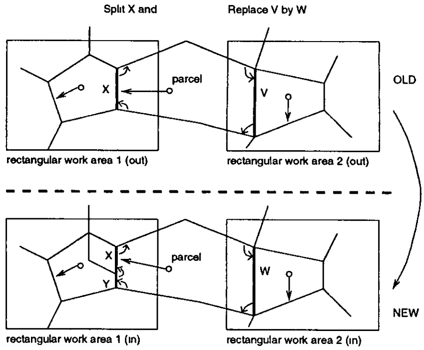

Fig. 5: Difficult check-in rectangular work areas

not to do so. An alternative solution is to mark error objects by setting an additional attribute `error_date`.

Another special case is the result of geometric data quality improvement. After obtaining new accurate reference points and 'rubber sheeting' related objects, many relatively small changes occur. It was decided to treat these as normal updates, because the customers must also have the same geometric base as the data provider. Otherwise, potential topological errors may occur (in the future) due to these small differences in the coordinates. However, the customers must be informed about quality improvement, because they will receive large update files.

4 Update files

As explained in the introduction, after an initial full delivery of the data set, the customers receive periodic update files, which contain the differences with respect to the previous delivery [16]. The time interval for a typical update file starts at the begin point in time `t_beg` and stops at the end point in time t_end. The update files are composed of two parts: OLD *(in Dutch* WAS*)*: deleted objects and old versions of changed objects; NEW *(in Dutch* WORDT*)*: new objects and new versions of changed objects.

Besides selecting these data from the database (using SQL queries with time stamps), the production of update files at least has to include reformatting the database output in the national data transfer standard NEN-1878 [17] or some other desired data transfer format. The object changes might occur in attributes, such as topological references, which the customer does not receive. These invisible changes can be either filtered out (*signif_changes*) or may be left in the update file (*all_changes*). There are two ways of interpreting the begin (t_beg) and end (t_end) time related to an update file: as a complete time *interval* or as two individual *points* (*instants*) in time. In the second case, the customer is not interested in temporary versions of the objects between the two points in time t_beg and t_end. This results in four different types of update files:

1. interval_all_changes: all changes over time interval (t_beg, t_end] including t_end, with delivery of all temporary object versions.

```
/* deleted/updated objects */
select * from line l where
  t_beg < l.tmax and l.tmax <= t_end;

/* new/updated objects */
select * from line l where
  t_beg < l.tmin and l.tmin <= t_end;
```

In case an object is updated two times, two versions of old objects (OLD: x,t1 and x,t2) and two versions of new objects (NEW: x,t2 and x,MAX_TIME) will be included in the update file; see the example below:

```
                                    oid=x,
                  oid=x,            tmax=MAX_TIME
oid=x,            tmax=t2       |------------->
tmax=t1       |-----------|
----------|                 t2
              t1
t_beg         (time line)          t_end
--O---------------------------X----->
```

2. points_all_changes: only changes comparing the two points in time t_beg and t_end, excluding all temporary versions, have to be delivered. This means that the object versions have to overlap in time either t_beg (deleted/updated objects) or t_end (new/updated objects).

```
/* deleted/updated objects */
select * from line l where
  t_beg < l.tmax and l.tmax <= t_end
                and l.tmin <= t_beg;

/* new/updated objects */
select * from line l where
  t_beg < l.tmin and l.tmin <= t_end
  and t_end < l.tmax;
```

In the example above this will produce only one version of the old object (OLD: x,t1) and only one version of the new object (NEW: x,MAX_TIME).

3. interval_signif_changes: all changes over time interval (t_beg, t_end] with respect to the delivered attributes (A1,A2,...,An) are included in the update file. Ai can be a geometric data type. As the data has to be reformatted anyhow by the front-end application in order to produce the standard transfer format NEN-1878, it is easy to include the filter for significant changes in this application (especially if the input data is sorted on `oid`):

```
select l.oid,l.tmax,l.A1,l.A2,...
from line l
where /* deleted/updated */
  t_beg < l.tmax and l.tmax <= t_end
  or /* new/updated */
  t_beg < l.tmin and l.tmin <= t_end
sort by l.oid, l.tmax;
```

4. points_signif_changes: all changes comparing the two points in time `t_beg` and `t_end` with respect to the delivered attributes `(A1,A2,...,An)` are included in the update file. It is now not true anymore that the reported object versions have to overlap in time either `t_beg` (deleted/updated objects) or `t_end` (new/updated objects), because they can be related to insignificant changes. It could be that a significant change occurs somewhere in the middle; see the example below:

```
                                  oid=y,
                        oid=y,   tmax=MAX_T
        oid=y,        tmax=t3 |--------->
oid=y,    tmax=t2 |--------|
tmax=t1 |--------|          t3
--------|         t2    insignif
        t1      signif   change
     insignif   change
      change
t_beg      (time line)       t_end
--O---------------------------X----->
```

In general, many insignificant versions of an object, w.r.t. the attributes for a customer, may precede and/or follow a version with a significant change. These should be temporarily glued together with versions related to insignificant changes; not in the database itself. This can be included easily in the application program in two steps: first 'glue', then filter out glued object versions, which do not overlap the two points in time: `t_beg` and `t_end`.

5 Future work

Visualizing changes over time requires implementing specific techniques [2, 12, 14] in a geographic query tool such as GEO++ [25]. The following is an overview of possible techniques to visualize spatial temporal data; more details can be found in [24]. *Double map:* Display besides each other the same region with the same object types but related to two different dates. *Change map:* Display the changed, new and deleted objects over a specified time interval on top of the map. *Temporal symbols:* Use a static map with thematic symbols for a temporal theme; e.g. depicting dates, change rates, order of occurrence, etc. *Space-time aggregation:* Aggregate the (number of) changed, new, and deleted objects to larger units in order to visualize the change rate in different regions. *Time animation:* Visualize changes through an animation by displaying the same region and object types starting at `t_beg` in n steps to `t_end`. *Time as third dimension:* Visualize changes over time, by using the third dimension for time. The user navigates through this 3D-space; see Fig. 6.

Although many aspects of maintaining topology and time in a database have been described, there are still

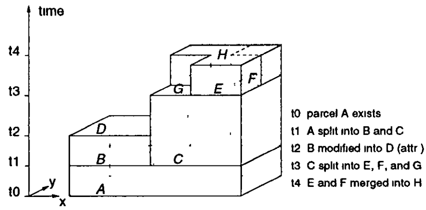

Fig. 6: 3D visualization of parcel changes over time

some open questions: 1. should we try to model the future?, and 2. how long should the history be kept inside the database tables? The current proposal is to keep the information in the database forever.

Returning to the first question: in addition to the history we might also want to model the (plans for the) future. In contrast to the past were there is only one time 'line', the future might consist of alternative time 'lines', each related to a different plan. There is a different type of 'time topology' for these future time lines; see [6]. In this case multiple versions are needed [5].

6 Conclusion

This paper shows how changes in map topology may be recorded in a temporal database by only using the `oid` part of the key for topology references and omitting the time part `tmax`. This avoids updating the neighbors in many cases. The check-in/check-out of workfiles enable long transactions and assure that the database is always in a correct state and that the spatial topology references are always correct. Further, the temporal topology is also correct as object versions are adjacent on the time line. The model allows 1. easy reconstruction of the situation for every given point in time, and 2. easy detection of all changes over a *time interval* or between two *points in time* for the production of several type of update files.

Acknowledgments
Many ideas with respect to storing topology and history were developed in early discussions with Chrit Lemmen. The developers of GEO++ (Tom Vijlbrief) and X-Fingis (Tapio Keisteri and Esa Mononen) were helpful with there comments. Paul Strooper, once again, thoroughly screened this paper, which caused a significant improvement. Finally, several colleagues (Berry van Osch, Harry Uitermark, Martin Salzman, Bart Maessen, Maarten Moolenaar, Peter Jansen, and Marcel van de Lustgraaf) volunteered to act as reviewers and all did give useful suggestions.

References

[1] C. Claramunt abd M. Theériault. Toward semantics for modelling spatio-temporal processes within gis. In *7th SDH*, volume 1, pages 2.27–2.43, August 1996.

[2] C. Armenakis. Mapping of spatio-temporal data in an active cartographic environment. *Geomatica*, 50(4):401–413, 1996.

[3] Bruce G. Baumgart. A polyhedron representation for computer vision. In *National Computer Conference*, pages 589–596, 1975.

[4] CA-OpenIngres. Object Magemenent Extention User's Guide, release 1.1. Technical report, June 1995.

[5] M. E. Easterfield, R. G. Newell, and D. G. Theriault. Version management in gis – applications and techniques. In *EGIS'90*, pages 288–297, April 1990.

[6] A. U. Frank. Qualitative temporal reasoning in GIS – ordered time scales. In *6th SDH*, pages 410–430, September 1994.

[7] C. M. Gold. An event-driven approach to spatio-temporal mapping. *Geomatica*, 50(4):415–424, 1996.

[8] C. S. Jensen, J. Clifford, and R. Elmasri. A consensus glossary of temporal database concepts. *SIGMOD Record*, 23(1):65–86, 1994.

[9] Kadaster, Directie Geodesie. Handboek LKI - extern, technische aspecten. Technical report, Dienst van het Kadaster en de Openbare Registers, November 1989. (In Dutch).

[10] Karttakeskus, Helsinki, Finland. Fingis User Manual, version 3.85. Technical report, 1994.

[11] T. Keisteri. Fingis – software and data manipulation. In *Auto Carto London*, volume 1, pages 69–75, September 1986.

[12] M. J. Kraak and A. M. MacEachren. Visualization of the temporal component of spatial data. In *6th SDH*, pages 391–409, September 1994.

[13] KT-Datacenter Ltd., Riihimäki, Finland. X-Fingis Software V1.1, INGRES version. Technical report, October 1994.

[14] G. Langran. *Time in Geographic Information Systems*. Taylor & Francis, London, 1992.

[15] C. Lemmen and P. van Oosterom. Efficient and automatic production of periodic updates of cadastral maps. In *JEC-GI'95*, pages 137–142, March 1995.

[16] C. H. J. Lemmen and B. Keizer. Levering van mutaties uit de LKI-gegevensbank. *Geodesia*, 35(6):265–269, September 1993. (In Dutch).

[17] NEN-1878. Automatische gegevensverwerking - Uitwisselingsformaat voor gegevens over de aan het aardoppervlak gerelateerde ruimtelijke objecten. Technical report, Nederlands Normalisatie-instituut, Juni 1993. (In Dutch).

[18] D. Peuquet and L. Qian. An integrated database design for temporal gis. In *Proceedings of the 7th International Symposium on Spatial Data Handling, Delft, The Netherlands*, pages 2.1–2.11, August 1996.

[19] D. J. Peuquet and E. Wentz. An approach for time-based analysis of spatio-temporal data. In *6th SDH*, pages 489–504, September 1994.

[20] H. Raafat, Z. Yang, and D. Gauthier. Relational spatial topologies for historical geographical information. *IJGIS*, 8(2):163–173, 1994.

[21] A. A. Roshannejad. *The Management of Spatio-Temporal Data in a National Geographic Information System*. PhD thesis, Enschede, The Netherlands, Twente Univeristy, 1996.

[22] R. T. Snodgrass, I. Ahn, and G. Ariav. Tsql2 language specification. *SIGMOD Record*, 23(1):65–86, 1994.

[23] M. Stonebraker and L. A. Rowe. The design of Postgres. *ACM SIGMOD*, 15(2):340–355, 1986.

[24] P. van Oosterom and B. Maessen. Geographic query tool. In *JEC-GI'97*, page ?, April 1997. To be published.

[25] T. Vijlbrief and P. van Oosterom. The GEO++ system: An extensible GIS. In *5th SDH*, pages 40–50, August 1992.

[26] A. Voigtmann, L. Becker, and K. H. Hinrichs. Temporal extensions for an object-oriented geo-data-model. In *7th SDH*, volume 2, pages 11A.25–11A.41, August 1996.

[27] M. F. Worboys. Unifying the spatial and temporal components of geographical information. In *6th SDH*, pages 505–517, September 1994.