Chapter 7

Constraints in Spatial Data Models, in a Dynamic Context

Peter van Oosterom
GIS-technology Section, Delft University of Technology, Delft, The Netherlands

7.1 Introduction

Constraints are important in every GI modelling process but until now have received only *ad hoc* treatment, depending on the application domain and the tools used. In a dynamic context, with constantly changing geo-information, constraints are very relevant; any changes arising should adhere to specified constraints, otherwise inconsistencies (data quality errors) will occur. In GIS, constraints are conditions that must always be valid for the model of interest. This chapter argues that constraints should be part of the object class definition, just as with other aspects of that definition, including attributes, methods and relationships. Furthermore, the implementation of constraints (whether at the front-end, database level or communication level) should be driven automatically by these constraints' specifications within the model. But, this is not possible yet, so this chapter will describe some implementation steps as interactively executed.

In certain applications some functions (linear programming in spatial decision support systems, survey least squares adjustment, cartographic generalisation, editing topologically structured data, etc.) partially support constraints. However, the constraints are not an integral part of the system and the constraint specification and implementation are often one and the same, and deep in the application's source code. The result is that the constraints are hidden in some subsystems (with other subsystems perhaps unaware of these constraints) and it may be very difficult to maintain the constraints in the event that changes are required. This is true for (G)IS in general, but is especially true for dynamic environments, with changing objects, where the support of constraints is required but presents a challenge. Example applications include cadastral or topographic data maintenance, Virtual Reality (VR) landscape design, and Web feature service.

7.1.1 Context

There are situations where certain types of constraints are well supported. Domain value constraints and referential integrity constraints in relational DBMSs (Date and Darwen, 1997) are standard functionalities. For example whenever one object refers to another via a foreign key, the DBMS checks that the referred object exits,

Dynamic and Mobile GIS: Investigating Changes in Space and Time. Edited by Jane Drummond, Roland Billen, Elsa João and David Forrest.

otherwise the transaction or change will not be committed. Another more specific GIS example is the support of topological constraints, such as certain types of objects, which may not overlap. Topological constraints can be supported within the DBMS, by, for example, LaserScan Radius topology (2003) and Oracle spatial 10g (2003) with topology, or they can be supported at the 'middleware' level such as in ESRI (2002). Within the context of VR systems, constraints are often implemented as the behaviour of objects. An illustration is the constraint 'two trees (objects) cannot grow on the same location', which is realised (hard coded in the edit environment) by collision detection, a well-known computer graphics technique. Referential integrity, topological correctness and collision detection are just a few examples of constraint types, but the available solutions may only work in certain subsystems. Other subsystems may not be aware of these and may have different 'opinions' of correct data. So constraints must be implemented at various levels (or subsystems), including application (edit, simulate,…) level, data exchange (communication) level and database level.

Although support for integrity constraints is patchy, there has been some research in this area. Primarily, integrity constraints are related to data quality (Hunter, 1996) and the source of errors (Collins and Smith, 1994) such as during data collection, data input, data storage, data manipulation, data output and the use of results. Cockcroft (1997) was one of the first researchers presenting a taxonomy of (spatial) integrity constraints. A contribution of the current chapter is a refinement of this taxonomy. Cockcroft (2004) advocated an integrated approach to handling integrity, based on a repository that contains the model together with the constraints. Cockcoft (2004) concluded that the constraints should be part of the object class definition, similar to other aspects of the definition. The repository is used both by the database and the application as a consistent source of integrity constraints. The current chapter continues these investigations into the possibilities of managing constraints in an integrated system-wide manner and adds data communication as an additional part of the system where constraints are important. It should be noted that much of the presented material is still a 'vision' and complete implementation is still in progress, though important parts have been proven.

7.1.2 Chapter overview

This chapter demonstrates the need for the integral support of constraints through four quite different cases: a VR system for landscape design (Section 7.2), cadastral data maintenance (Section 7.3), topographic data maintenance (Section 7.4) and a Web feature service (Section 7.5). All four applications deal with dynamic situations. The landscape design has an explicit temporal aspect, namely the simulation of tree growth. During both the initial design and the simulation these constraints should be met. In the case of the cadastral application, when parcels are changed, constraints have to be satisfied otherwise this could lead to inconsistencies, such as parcels overlapping or lacking an owner. Not further discussed, is in-car navigation using a topographic base map: if the moving point belongs to a car, a constraint could be that the point should always be on, or near, a road or related features, such as a parking lot. Based on the different constraints,

experiences in the four cases (and the relevant literature), a classification of constraints is given in Section 7.6. Constraints can be related to the properties of an object itself and can also be based on relationships between objects. Constraints such as ‘a tree must always be green’ or ‘the salary of a staff member should be higher or equal to the minimum salary’ illustrate constraints based on properties of only one object. Examples of constraints considering relationships between two objects are ‘a Yucca tree must never stand in water’ and ‘the salary of the boss must always be higher than the salaries of the other staff’. These constraints require formal description and definition. Section 7.7 discusses the formal specification of constraints within a (conceptual) model. The implementation of constraints, with focus on the DBMS, is described in Section 7.8. This chapter’s last section concludes with the principal results and proposes further research directions.

7.2 Constraints in a landscape design VR system

With SALIX-2 (van Lammeren et al., 2002) a user can interactively introduce new objects (trees, bushes, etc.) to a 3D landscape. As is the case in reality, sometimes new objects have to be a certain distance from each other (for example, two trees have to be planted not closer that 3 m), from other objects or are even not in an area at all, for example a tree on a road (Louwsma, 2004).

7.2.1 SALIX background

Digitally supported landscape design contains intriguing challenges. These challenges have to do with modelling the changes in time of the architectural primitives (mainly trees and shrubs) and modelling the relation between architectural objects, their architectural primitives and their spatial configuration. Virtual Reality (VR) tools such as VR-construction sets and VR-viewers are widely available, and provide opportunities to experiment with a wide range of design proposals using a geo-database representation. Such (VR) geo-information systems offer a three-dimensional laboratory to experiment with landscape design proposals. SALIX-2 is a simulation program, exploiting these possibilities, developed for students of landscape architecture at Wageningen University (van Lammeren et al., 2002) (see Figure 7.1).

VR-scene manipulations make it possible to interact with a virtual scene object (Heim, 1998) such that an object (or its attributes) in the scene can be deleted or added. With SALIX-2 the underlying idea is a virtual environment for simulating the growth of plantation objects (bushes and trees). The students are able to plant bushes and trees interactively. Just as in the real world, one should be restricted from planting in particular areas. For that reason the system has to be provided with constraints related to the type of plantation and geo-information objects.

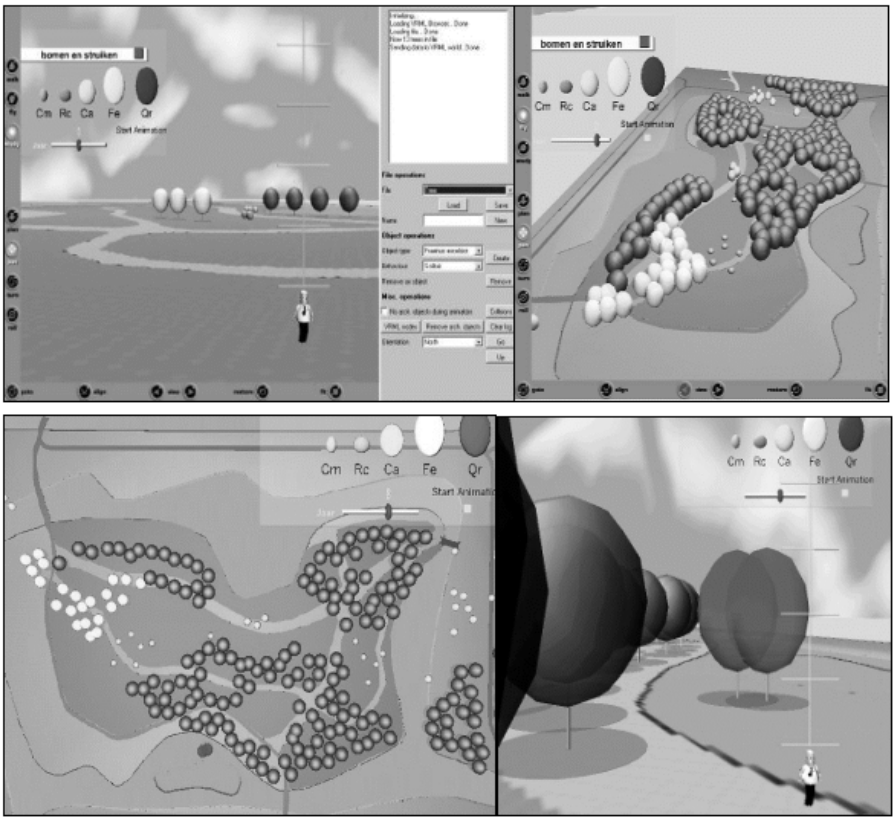

Figure 7.1. 3D scenes of SALIX-2: an interactive landscape modelling system. Lower right part: A constraint is violated in SALIX-2c (note the red, highlighted trees on colour version following page XXX).

7.2.2 Selected constraint examples

The SALIX-2 system currently maintains three classes of objects: trees, bushes and ground surfaces. The possible ground surfaces are water, paving, soft_paving, grass and bridge. There are five possible types of trees/bushes (CorAve, CorMAs, FraxExc, QueRob, RosCAn). Examples of rules for the position of objects in geo-VR environments can be: a tree must not overlap with water or a tree must be

covered by a polygon with destination forest. For these constraints it is logical to represent a tree as a circle (an extended object) and not as a point (centroid). Table 7.1 shows examples of constraints for SALIX-2; see also Figure 7.2.

Table 7.1: Selected examples of relationship constraints for SALIX-2.

Type of relation	**Constraints formulated with forced relations between objects**
Direction	A bush always has to be placed south of a tree
Topology	Bushes always have to be disjoint or meet water A bush always has to meet or be disjoint with paved areas (also thematic constraint) (2 predicates)
Metric	Trees always have to be positioned > 1 metre from paving
Temporal	An oak always grows for 70 years
Quantity/ Aggregate (sum)	There must always be at least 10 trees on the specified ground surface
Thematic	A bush always has to meet or be disjoint with paved areas (note the mixed topological constraint)
Complex	The distance between trees inside water always is > 8 m AND the distance between the tree and the edge of the water always has to be < 0.5 metre AND the species must be a salix

7.2.3 Some lessons

The main lessons learnt with respect to the constraint support requirements of SALIX-2, the VR landscape modelling system are (Louwsma, 2004):

- constraints occur at different places, both in the VR user interface and data storage;
- when designing, immediate feedback to the user is important (see Figure 7.1, bottom); and
- simulation adds another ‘dimension’ to constraints, when creating an initial plantation layout everything may be correct, but after 5 years of simulated growth there may be conflicts, e.g. trees get too close.

7.3 Constraints in a cadastral application

In this section a cadastral data maintenance system (another application in which constraints play a major role) is discussed. Although cadastral systems also maintain important legal and administrative information, this section’s focus is the spatial side of cadastre.

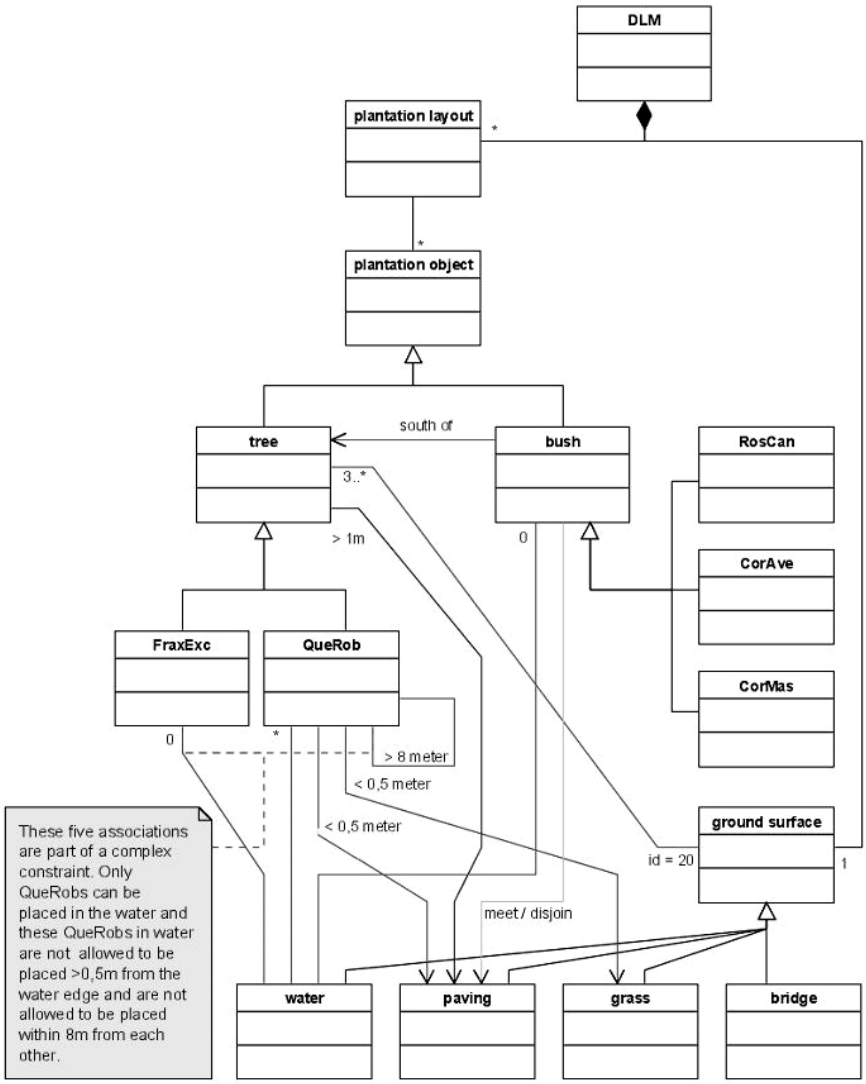

Figure 7.2. UML classs diagram representing the objects of intrest and their constraints in SALIX-2 (see colour insert following page XXX).

7.3.1 Dutch cadastral data

The Dutch cadastral map is based on a winged-edge topology structure (Van Oosterom and Lemmen, 2001); see Figure 7.3. The DBMS is considered very clean,

topologically. Further, the model contains redundancy in the topological references: both the (meaningless system) object_id reference to the left and right parcels and the (meaningful user) parcel_number references to the left and right parcels are stored and maintained. The topological consistency checks are hard coded and built into both the editor and the check-in software at the DBMS server side. However, the checks are currently not implemented within the DBMS itself (Ingres). The data set covers the Netherlands and contains history from 1997 to the present. The total number of current boundaries (polylines) is about 22,000,000 and the number of current parcels (topological faces) about 7,000,000. If all historic versions are counted, numbers roughly quadruple. There is a separate, but linked, subsystem containing the legal and administrative data.

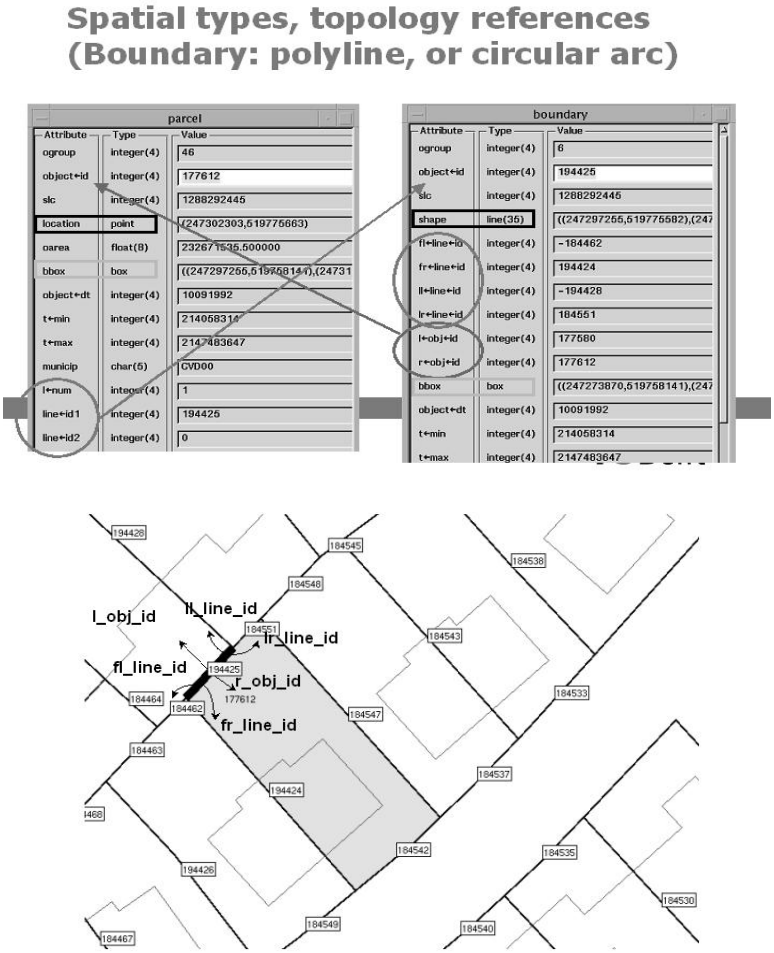

Figure 7.3. Winged-edge topology structure of the cadastral map.

7.3.2 Some examples of cadastral data constraints

Due to redundancies in the system and because, in general, topology references can be derived from the metric information, a large number of consistency checks can be defined for the cadastral model. Over 50 constraints have been defined, in a number of different categories. In this section some example categories will be presented, accompanied by SQL select statements, which in the case of correct data should not find any objects. These statements could be considered the body of SQL assertions, with the 'create assertion' part skipped (see Section 7.8). (Discussion of constraints related to attribute value domain checks are also skipped, being trivial.) Five categories of cadastral constraints will be discussed.

1. *Metric checks.* The first example finds closed 'arcs' (but not circles), which can be detected by checking that the first and last (third) point defining the arc match, see CCVQ1 (Appendix 1 with the Cadastral Constraint Violation Queries). A second example constraint disallows straight 'arcs' (see Figure 7.4). Another example ensures every parcel has a reference point, which should be within the area of the parcel; this reference point should also be in the bounding box of the parcel, which is easily checked with the CCVQ2. The final example is that two different boundaries should not intersect, but should be disjoint or touch at their end points.
2. *Existence of topological references*. This can be compared to referential integrity checks in some administrative databases. A complication is that topological references can be signed (+ or -) in order to indicate proper orientation. The first constraint in this category checks whether the left (and right) parcel references from the boundaries do indeed exist (CCVQ3). The next example checks whether the winged-edge boundary-boundary reference (in this case the first left references) exists (CCVQ4). Then, starting from the parcel, a number of topological reference checks can be imagined. For example, as parcels can have island boundaries, these references also have to be correct. So, the reference from the parcel to the island boundary reference must exist. Further, as a parcel can have any number of islands (and the number of islands is encoded as an explicit attribute), it must be checked whether the correct number of parcel references are specified and if they all refer to existing boundaries. The final example in this constraint category checks whether the reference from the parcel to its outer boundary exists (CCVQ5).

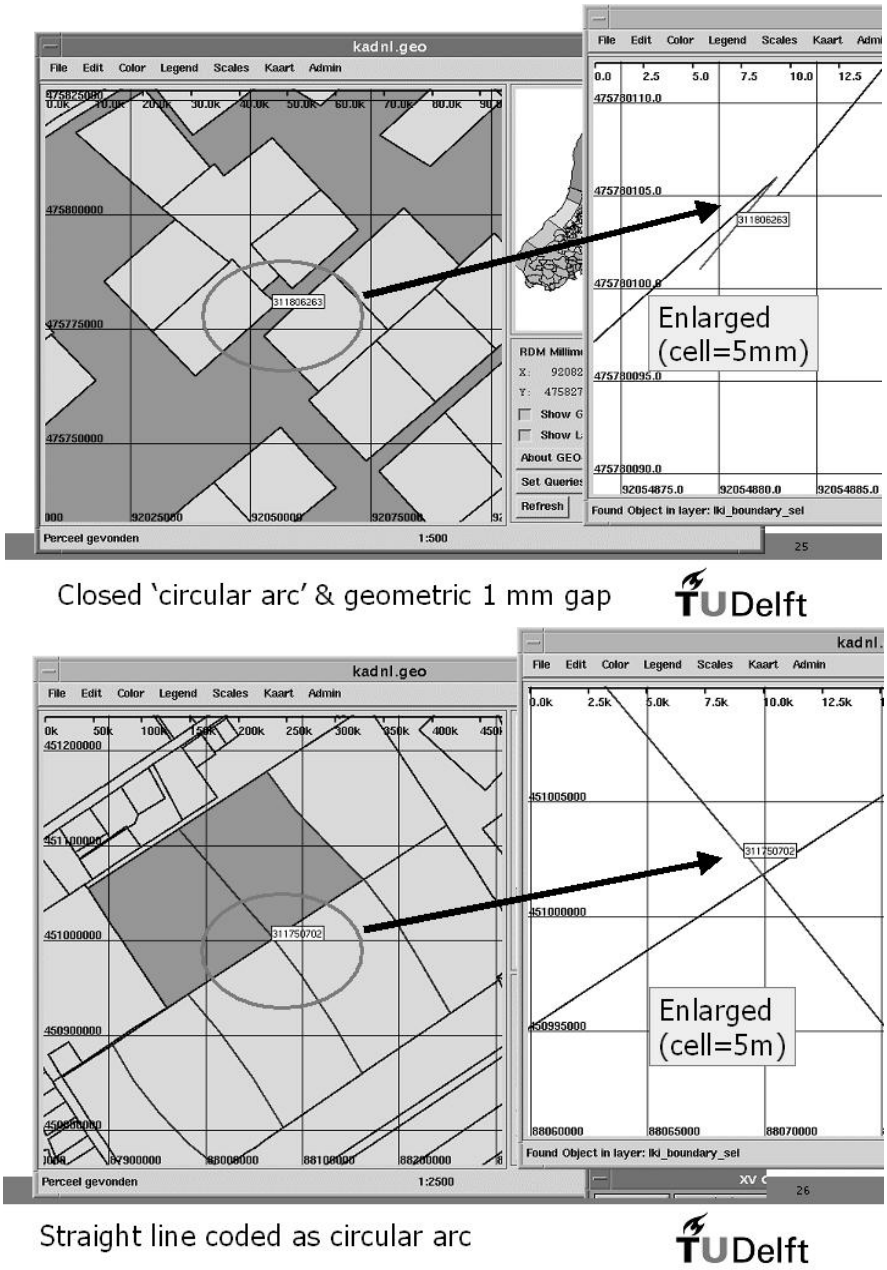

Figure 7.4. Some metric errors in the cadastral dataset (top: small gap between two boundaries, bottom: straight line encoded as circular arc).

3. The *correctness of a topological reference*, see Figure 7.5. A first example in this category is the check that two consecutive boundaries must have the same parcels on one side. In total there are eight combinations that have to be checked as each of the four winged-edge boundary-boundary references is signed, that is, the direction of the next edge may have to reversed (thereby switching the left and right hand sides). CCVQ6 checks the positive first left reference. A similar constraint in this category is that the end point of one boundary is the start of the next. As with the previous consistency check, there are again eight combinations which have to be checked; CCVQ7 shows the positive first left case again. Also in this category of constraints is the check as to whether the island boundary has the parcel at the correct side. Another constraint is whether the first coordinate of the island boundary lies within the bounding box of the parcel. Finally a check is given to see whether the outer boundary and parcel references back-and-forth are consistent (CCVQ8).
4. The fourth category of constraints to be considered is a *referential integrity check*, which determines whether two subsystems are consistent. The two subsystems are the geometric subsystem (LKI) and the administrative and legal subsystem (AKR). Every ground parcel in AKR should also be present in LKI (CCVQ9).
5. *Temporal constraints* ensure that the time intervals of two consecutive versions of an object do touch and assume no gaps or overlaps in the time dimensions of an object.

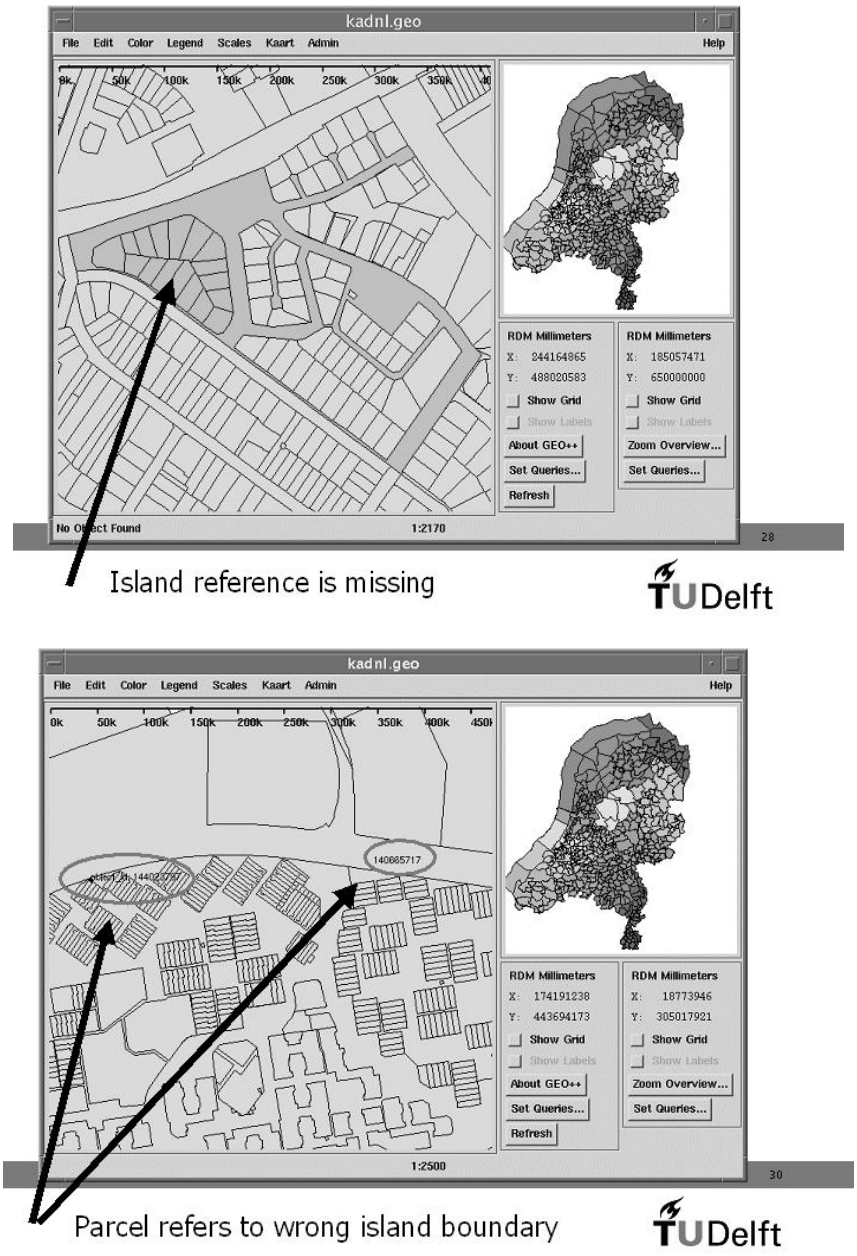

Figure 7.5. Some topology reference errors in the cadastral dataset (top: island reference is missing, bottom: parcel refers to wrong island).

7.3.3 Some lessons from cadastral data

The cadastral dataset is considered clean and is, by the nature of its structure, designed to avoid certain errors, such as overlapping parcels. During the conversion in 1977 from the old to the new version all consistency errors were resolved and removed. Further, the cadastral data has been delivered to many different customers and loaded into different systems, each of which is potentially sensitive to different errors. As the customers pay for the data, they will complain quickly about errors. However as the constraints discussed in the previous section and applied to the production data of 2004 have clearly illustrated, certain errors have, despite everything, been (re-)introduced (see van Oosterom et al. [2005] for more details). One important lesson has been that one should trust neither front-end nor middle ware alone for consistency checking, but implement checking throughout the whole system and particularly within the DBMS, which will contain what is considered valid data shared by multiple users. Further, even if the errors are not noticed in the production environment, they may be harmful in the users' environments; e.g. straight 'circular arcs'. Despite a thorough treatment of the different categories of constraints, not all possible constraints have been discussed. In the category of topological correctness, for example, it is not considered whether the complete domain is covered with parcels. This is an important type of topological constraint as there should be no gaps – in the cadastral case this is equivalent to an area without an owner.

7.4 Constraints in a topographic application

At first sight the types of constraints relevant to cadastral update and topographic update systems seem similar. But it has been decided to include a topographic application in this chapter's cases. The reason is that currently the Dutch topographic data maintenance system is being completely redesigned. The new design contains constraints within the specifications of the data model. Besides renewing the production environment (including a move from separate files to one geo-DBMS), the product itself is being renewed as TOP10NL. It is more object-oriented than map-oriented, contains nationwide unique identifiers and will be delivered in GML-3 from January 2006.

7.4.1 Constraints in the TOP10NL

The constraints were initially designed for the conversion process, to make sure the new topographic production system only contained clean data. However, the same constraints will be used during future production editing (and this has been successfully tested in prototype versions of the future system) and geo-DBMS check-in. The constraints are specified in one source, which contains the complete model (or specification) of all object classes, attributes and relationships. The model is encoded in an XML-format developed by Vertis and the Topographic Service. In the remainder of this subsection fragments from this XML-format will be shown to illustrate a number of example constraints. The following five types of constraints were recognised (using Vertis/Topographic Service terminology):

1. Single entity, single attribute (thematic). This example shows a domain constraint specifying the fact that the width of water should be between one and 500 (metres) and the same XML encoding of this range domain type states that the default value is six (metres); see TMCX1 (Appendix 2 Topographic Model Constraints in XML). In the same category is the specification of valid values of the XML encoding for the railroad enumeration type, with allowed values 'verbinding' (connection) and 'kruising' (junction) given in TMCX2.

2. Single entity, multiple attributes (thematic). An example from this category checks a road object constraint that the 'NAAM_AWEG' (name a-road) attribute is filled and then the attribute 'WEGTYPE' (road type) must contain a specific value; in this case 'autosnelweg' (highway). Note the specific operator 'MVCONTAINS' which is used in the case when a multi-valued attribute contains specific value(s). The corresponding XML fragment is given in TMCX3.

3. Geometry (general rules, including minimum line length and minimum area). For example, if the width of a road is less than two metres, then the geometry type is line, otherwise the geometry type is area. The example from this category will not be further illustrated here in XML, because it is of the same type as the first category except that a constraint is specified for a geometric, not thematic, attribute.

4. Topology (several subtypes: covering_without_gaps, no_overlap, coincide, …) As the model of the Topographic Service is not based on a topological structure, but consists of individual point, polyline and polygon features, topology constraints look different from those used for the cadastral data set, which was based on a topological structure. One could even state, in this case, that constraints are even more important, because there is no other facility supporting topological data quality. TMCX4 shows the constraint that two roads 'WEG_VLAK' (road area) may not overlap at the same height 'HOOGTENIVEAU' (height level). (Note the use of the topology operation 'AREAOVERLAPAREA' from the ESRI ArcGIS environment.)

5. Relationship. Every feature must have at least one specified source. The example TMCX5 checks that the return value of the operator 'BRONCOUNT' (source count) is greater than '0'.

7.4.2 Some lessons from the new topographic base data production system

The fact that the constraints are specified together with the model and used as a source for the realisation of different (sub)systems is a great leap forward. The same constraints are applied during the initial conversion from the old to the new system, during interactive editing (ArcGIS environment; see Figure 7.6) and at data storage (Oracle DBMS) during check-in.

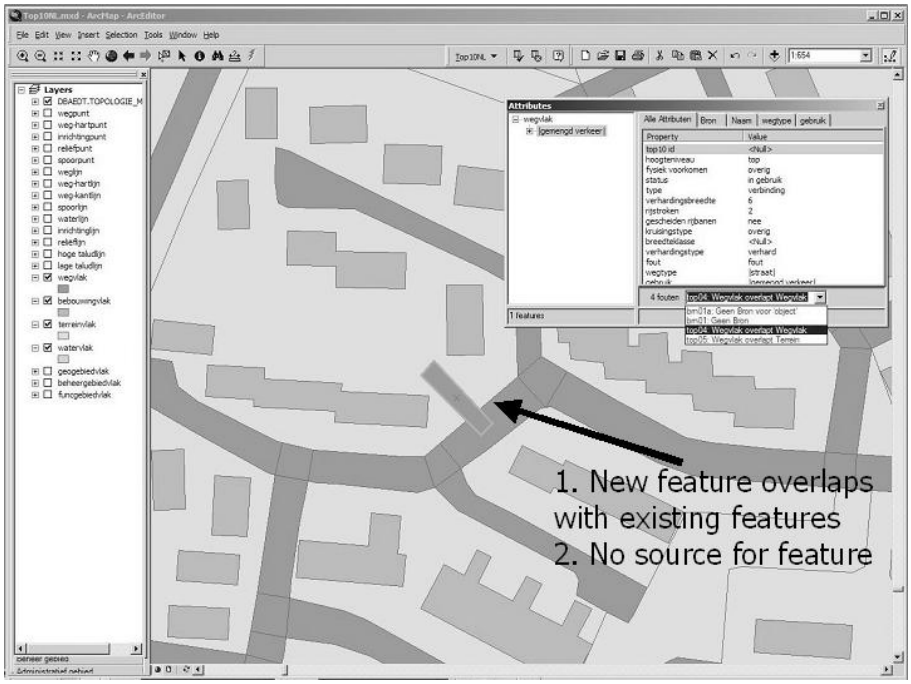

Figure 7.6. An error caught during editing of the topographic data.

Of course, things can always be further improved, for example:

1. the model itself could be specified in UML (and based on OGC/ISO TC211 standards);
2. instead of institutionally generated (XML) constraint encoding, perhaps a standard would have been better, e.g. OCL;
3. the exchange format is not (automatically) derived from the same source model (as used for the edit and storage subsystems); and,
4. the constraints are not yet included in the exchange format. Some could possibly be included in standard GML/XML encoding (for example the domains constraints) but more research is needed for the other types of constraints.

7.5 Constraints in a Web feature service

The last case to be presented also relates to the cadastral domain, but the context is quite different: an Internet GIS environment. This different context will reveal new insights into the important role constraints play in real-world implementations, and the current difficulties experienced in supporting them. With the availability of the standard OpenGIS Web Feature Service (WFS) (OGC, 2002) protocol, it is now finally possible to realise Internet-based geo-information processing environments,

where clients cannot only view but also edit (update) data stored at multiple servers, and where products of one vendor can be combined with the server products of another. An evaluation of the WFS-Transaction protocol was carried out using a case study known as 'notary drafts cadastral parcel boundary'. The WFS protocol was analysed (Brentjens, 2004; Brentjens et al., 2004) and revealed, for more advanced edit scenarios, a number of possible improvements related to constraints.

7.5.1 Web feature services

Two classes of Web Feature Services are defined in the OpenGIS WFS specification: Basic WFS (for retrieving geographic features) and Transaction WFS (needed for editing geo-data) (OGC, 2002). The WFS protocol allows a client to retrieve geo-spatial (vector-) data encoded in Geography Markup Language (GML) from multiple Web Feature Services. GML is an XML encoding for the modelling, transport and storage of geographic information, including both the spatial and non-spatial properties of geographic features (OGC, 2003).

A 'Basic WFS' service implements the GetCapabilities, DescribeFeatureType and GetFeature requests. A client can request an XML-encoded capabilities document (containing the names of feature types that can be accessed via this service, the spatial reference system(s) and the spatial extent of the data, plus information about the operations that are supported) by sending the GetCapabilities request to the WFS service. The function of the DescribeFeatureType request is to generate an XML Schema document (XSD, 2004) with a description of the data structure of the feature types serviced by that WFS service. The GetFeature request allows for the retrieval of feature instances (with all or part of their attributes).

A 'Transactional WFS' offers the functionality for modifying geographic features as well, such as insert, update and delete. In order to do so, a Transactional WFS implements the Transaction request (a set of insert, delete and update actions that belong together). It could optionally implement the LockFeature and the GetFeatureWithLock request. When the transaction has been completed, a WFS will generate an XML response indicating the completion status of the transaction (and a list of newly generated feature identifiers assigned to the new feature instances). The purpose of the LockFeature request is to invoke a mechanism ensuring consistency by preventing simultaneous editing of these features by other users. A Lock element uses a filter to specify which feature instances should be locked. Finally, by using the GetFeatureWithLock instead of the GetFeature request, a client requests features be retrieved and locked simultaneously.

7.5.2 Notary drafts new parcel boundary

An example cadastral transaction is a notary who sketches a new parcel boundary as a parcel is split following a property transaction. The functional requirements for the interface between the Cadastral WFS server and the notary consist of the following requests (see Figure 7.7 top right): 1. Transfer whole parcel, 2. Merge two or more parcels, 3. Split parcel, 4. Get a reference cadastral map. The split of a parcel is the most interesting case as the required input is a parcel number; the result is a GML dataset with the parcel and context (if parcel does exist). The client action

is to add one (or more) parcel boundary within the area of the parcel to be split and thereafter every implied part of the parcel is uniquely labelled with a text string (usually one letter ‘A’, ‘B’,..), (see Figure 7.7, bottom). Below is an example of the WFS protocol to pose a request to get two types of features (with the DescribeFeatureType request):

```
http://130.161.150.109:8080/geoserver/wfs/wfs?request=
DescribeFeatureType&typeName=DRAFT_BOUNDARY,DRAFT_PARCEL
```

Below the XML/GML fragment with geo-information is sent from the server to the client (or, in the case of an update, in the other direction):

```
<gml:MultiLineString
 srsName="http://www.opengis.net/gml/srs/epsg.xml#28992">
 <gml:lineStringMember>
   <gml:LineString>
     <gml:coordinates decimal="." cs="," ts=" ">
       106417204,448719275 106367184,448675614
     </gml:coordinates>
   </gml:LineString>
 </gml:lineStringMember>
</gml:MultiLineString>
```

7.5.3 Evaluation of Transactional WFS

Though ‘simple’ editing proceeds well, a number of observations related to editing complex situations, such as handling topology (van Oosterom, 1997), can be made. First, concerning the integrity constraints in transactions, validation of (changes in) features should prevent a dataset containing features that violate topological rules or other restrictions. The WFS specification defines some operations and mechanisms that can be used for the validation of single features. It is not so easy however to enforce integrity constraints that concern combinations of features (as in the case of topologically structured data) or constraints that follow business rules. The server may check certain integrity constraints after the client posts a transaction and as a result the transaction may fail. However, the client does not know what the constraints are (except for the conditions implied by the GML application schema defining the individual feature types). It may be quite frustrating for a client to update data and then receive errors. An ongoing research topic is how and where to check integrity constraints (and other business rules) in WFS-based distributed systems.

One interesting question is whether it is possible (and meaningful) to translate constraints in the data model to constraints related to the structure of valid transactions. For example, a parcel split always implies at least deleting one old parcel, inserting a new boundary and two new parcel reference points on each side of the new boundary. How should constraints be related to operations in a valid transaction?

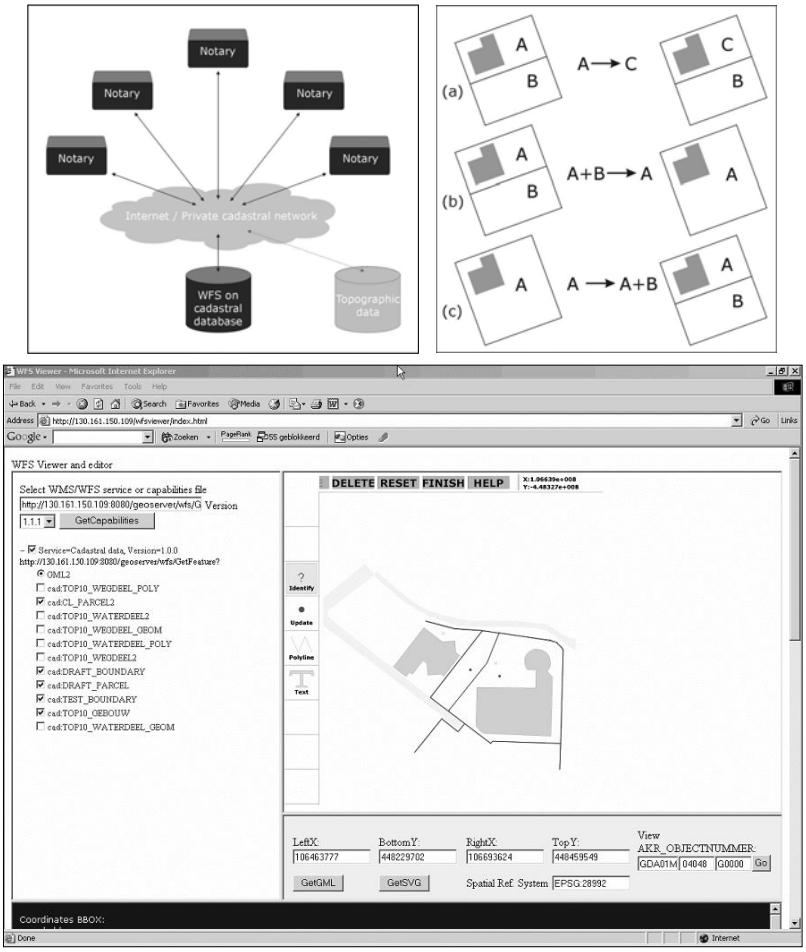

Figure 7.7. Case 'notary edits cadastral map via Internet' (top left: architecture of the Web feature server, top right: a number of typical cadastral edit operations, bottom: the edit Internet WFS client).

7.6 Classification of constraints

In order to better understand constraints and their use, it is important to classify them, including their spatial/dimensional aspects. Classification of the different types of (spatial) constraints reveals a complex taxonomy. Cockcroft (1997) presents a two-dimensional taxonomy of (spatial) constraints: the first axis is the static versus transitional (dynamic) distinction and the second axis is the classification into topological, semantic and user constraints. It is recognised that the transitional aspect of integrity constraints (allowed and valid operations; see also Section 7.5) is relevant, but in this chapter this is considered to be the 'other side of

the same coin'. This section refines, based on the four presented cases, the second axis of Cockcroft's taxonomy by recognising five subaxes (or five different criteria) for the classification of integrity constraints:

1. the *number of involved objects*/classes/instances;
2. the *type of properties of objects and relationships* between objects involved: topologic (neighbourhood or containment), metric (distance or angle between objects), temporal, thematic or mixed;
3. the *dimension* (related to the previous axis): 2D, 3D or mixed time and space, that is, 4D;
4. the *manner of expression*: 'never may' (bush never may stand in water) or 'always must' (tree always must be planted in open soil); and
5. the *nature of the constraint* can be: 'physically impossible' (tree cannot float in the air) or 'design objective' (bush should be south of tree).

With respect to the first subaxis of the constraint taxonomy, 'the number of involved objects', the following cases can be identified:

1. one instance (restrictions on attribute values of a single instance);
2. two instances from the same class (binary relationship);
3. multiple instances of the same class (aggregate);
4. two instances from two different classes (binary relationship); or
5. multiple instances from different classes (aggregate).

Further, the fourth subaxis, 'the manner of expression' has only practical value for communicating the constraints between the users. Once the objects and the constraints are formally defined, the expressions 'never may' and 'always must' can be represented by one constraint, e.g. by the one that is more efficient from an implementation point of view. For example, the constraint 'a tree never may stand in water, or street or house' is equivalent to the constraint 'a tree always must be in a garden, or park' under the assumption that there are only five possible ground objects: water, street, house, garden and park.

Further consideration will be given to the second subaxis of the constraint taxonomy, that is constraints with respect to object attributes, spatial relationships and dimensions. A distinction is made between constraints:

1. related to properties (attributes) or the state of objects, whether thematic, temporal or spatial (see subsection 7.6.1); and
2. based on (spatial) relationships between objects (see subsection 7.6.2).

7.6.1 Constraints derived from the properties of objects

The categories of constraints are: thematic (non-spatial business logic-like), temporal, spatial (area, perimeter, length) and also mixed. *Thematic* information about objects can be found in the attributes of the objects, such as house (number of

floors), road (maximum speed), grass (type). Thematic constraints ensure related attributes only get allowed values. *Temporal* property constraints specify allowed values for one or more of the time attributes; for example the start time (birth) of an object should be before the end time (death) of that same object. Other temporal property constraints may specify some valid values for time attributes; e.g. the date/time associated with an historic fact, should be somewhere in the past. *Spatial* constraints can be associated with spatial properties of one object such as size or shape. An example of size constraint can be 'a tree should not become higher than 30m', 'a canal must be at least 2 metres wide', and a constraint on shape could be 'a bush must be represented with a sphere'.

7.6.2 Constraints derived from spatial relationships between objects

The proposed (sub)classification of integrity constraints based on relationships between objects has the following components:

1. *thematic*, 'a parcel must always be owned by at least one person'. These relationship constraints are similar to the constraints for relationships found in business logic for non spatiotemporal systems.
2. *temporal*, 'the second object may only occur after the first object (adjacent in time)'. These relationship constraints are to be specified on the basis of frameworks for describing temporal relationships between objects. Peuquet (1995) and Kwon et al. (1999) describe the temporal relations between two time intervals. Given two time intervals, there are seven distinct ways in which these time intervals can be related (e.g. Before, Meets, Overlaps, Finishes, During, Starts, Equals). The temporal relations can be seen as relations between two objects with some time interval as existence time (with start and end time of existence as the boundaries of the time interval).
3. *spatial*. The formalisation of spatial relationship constraints is closely related to the formalisation of (spatial) associations between the objects. Thus when defining constraints the following subtypes exist:
   a. *topology* 'no trees and bushes inside water polygons; no trees and bushes inside paving polygons'. Topological constraints are to be constructed using frameworks for neighbourhoods (Egenhofer, 1989; Clementini et al., 1993). For example, if the boundaries of the two objects intersect but the interiors do not, the conclusion is that the objects meet. The constraint 'no bush in water' can be translated to 'no point-in-polygon' (assuming that a bush is represented as a point and water as a polygon), corresponding to the topology relationship 'not inside'.
   b. *direction* 'the trees should always be north of paving polygons, so people can walk in the sunshine'. Direction constraints are to be based on formalism for directional relations (Papadias and Theodoridis, 1997). The directional relations are defined as the position of an object with respect to another object, as the directions can be given in degrees in the range of [0°,

360°] or verbally (Northeast, North, Northwest, West, Southwest, South, Southeast, East) with each expression standing for an interval of degrees. Algorithms are also developed to assign the right direction to an object.

c. *distance* 'no trees inside the water, except if < 1 metre from edge or water bushes > 1 metre from paving (so the leaves do not overlap the paving)'. Distance constraints impose a constraint on a (Euclidian) distance between objects. They can be expressed in linear metres, or by more approximate linguistic terms such as 'closer than', 'further than' or 'interval distance'.

4. *mixed*, such as quantity (or aggregate): 'the maximum number of 10 plantation objects in a specified area in the centre of the park'. It is common to mix these fundamental types of relationship constraints. Specifying a certain density of objects in a certain area is only implicitly related to spatial relationships. Knowing the distribution of objects in a certain area, the minimum distance between two objects can be computed and, eventually, the approach for metric constraints can be used. From a user point of view, however, a more intuitive approach will to specify a number per given area. This constraint can be given as a minimum, exact or maximum number of objects for an area (surface density). Examples of density constraints are 'maximum number of houses in a residential area' or 'minimum number of trees in an area'. Examples can be 'one tower, three benches and one statue must be placed in this garden'. This exact number of certain objects can be seen as a special case of a density constraint, because it can be defined as an exact number of certain objects for the whole area in the 3D model.

7.6.3 Dimensional aspects of constraints

The last aspect to be discussed here is related to dimension. In general all spatial relationship constraints can be specified for both 2D and 3D objects. Constraints can concern the 2D ground plane or the 3D objects (bushes and trees) that could be placed on the ground plane to create a spatial configuration. The rules concerning the ground plane find their origin in local policy and in the fact that some designations conflict with each other. The policy makers define area designations and note them in plans. For example, some areas get an urban designation, some rural and others are agricultural. These designations of the ground plane can easily be stored as an additional attribute of the separate polygons. However the policy makers' defined restrictions could still conflict: e.g. a road can never lie in water (except when a bridge or tunnel is built), a forest never lies on a major road and all houses should be reachable by a path or road. On the other hand, such conflicting constraints could be a source for strategic spatial decision making; and the rules for 3D objects can even be more complex, too.

7.7 Specifying constraints

Having seen the importance of constraints in different applications and presented a refined taxonomy, the next issue is how to specify the constraints. First of all, the

specification of the constraints has to be intuitive for the user. The constraints have to be included in the object model. This model should be as formal as possible to be able to derive constraint implementations within the different subsystems (edit, store, exchange). Formal modelling is an essential part of every large project, but it is also helpful in small and medium-sized projects. Using a formal model permits communicating ideas with other professionals as well as describing clear, unambiguous views on implementation strategies. The Unified Modelling Language (UML), now a more or less 'default' state-of-the-art approach, will be used for object-oriented modelling (OMG, 2005a, Ch. 3). UML is a graphic language, which gives a wide range of possibilities for representing objects and their relationships. In general the language can be used for modelling business processes, classes, objects and components, as well as for distribution and deployment modelling. UML consists of diagram elements (icons, symbols, paths, strings), which can be used in nine different types of diagrams. The most appropriate diagram is the class diagram. It provides formalism for describing the objects/classes, with their attributes and behaviour, and relationships between these objects, such as association, generalisation and aggregation.

Despite their potential for formalizing objects and processes, UML class diagrams are typically not sufficiently refined to provide all the relevant aspects of constraints. Constraints are often initially described in natural language. Practice has shown that this results in ambiguities. In order to write unambiguous constraints, a non-graphic language is provided within UML for the further modelling of semantics (knowledge frameworks), namely the Object Constraint Language (OCL) (OMG, 2005b, Ch. 6). When an OCL expression is evaluated, it simply returns a binary value. The state of the system will not change when the evaluation of an OCL expression returns false. The advantage of using OCL is that – as with UML class diagrams – generic tools are available to support OCL (i.e. it is not GIS-specific); OCL has been used successfully in the context of GIS, an example is the IntesaGIS project with the GeoUML model specifying the 'core' geographic database for Italy (Belussi et al., 2004). The context of an invariant is specified by the relevant class; e.g. the object class 'parcel' is the context of the constraint 'the area of a parcel is at least 5 m^2. It is also possible within a constraint to use the association between two classes (e.g. every instance of the object class 'parcel' must have at least one owner, which could be depicted as an association with the class 'person'). OCL enables one to formally describe expressions and constraints in object-oriented models and other object modelling artefacts. Below are two examples in UML/OCL syntax (keywords in bold print):

```
context Parcel inv minimalArea:
     self.area > 5

context Parcel inv hasOwner:
     self.Owner -> notEmpty()
```

Figure 7.2 shows the UML class diagram with the objects and the constraints (depicted as associations) used for SALIX-2 (introduced in Section 7.2 and Table

7.1). In principle there is no difference between a ‘data model’ relationship (association, aggregation, specialization) and a ‘data model’ integrity relationship constraint. Both are depicted as lines in the UML class diagram. From a high level conceptual (philosophical) point of view the difference may be very small. However, normal associations are often indented, in subsequent implementations, to be explicitly stored (in one or both directions), while the relationship constraints should not result in such an explicit storage, but in a consistency rule in the implementation environment. In order to make a difference between the two, normal relationships are depicted in black while integrity relationship constraints are depicted in colour. In the diagram notes can be used to explain the constraints on relationships and/or properties. These notes can contain either UML/OCL or natural language text.

7.8 Implementation of constraints

The specified UML (OCL) models (including the constraints), managed in a repository, should be the foundation for all subsystems, including the edit/simulation environment, the storage database (further described in the DDLs of the DBMS) and the data exchange subsystems. The edit/simulation application subsystem will not be discussed here. However, it is considered important to incorporate the integrity constraints from the model (automatically) in the applications (as has been done in the topographic application; see Section 7.4). With respect to data exchange, the eXtensible Markup Language (XML) can be used for the models containing the class descriptions at class level (XML schema document ‘xsd’) and for the data at object instance level (‘normal’ XML document with data ‘xml’). XML documents also include the geometric aspect of objects (e.g. LandXML, GML, X3D). Further investigations are needed to incorporate integrity constraints in the XML schemas. The UML models (incl. the OCL) with constraints should be automatically translated to XML schemas. Note that this is different to encoding a UML model in an XML document according to the XML Metadata Interchange (XMI). In this section the implementation of integrity constraints will be illustrated with database examples. When the data are correctly stored in the DBMS all users will have access to the same consistent dataset.

Since SQL92 ‘general constraints’ (assertions) are part of the standard and could be used to implement the OCL constraints. However, assertions are not supported in the currently available DBMSs and developers are referred to the use of triggers and procedures. Assertions may be considered as an intermediate step between UML/OCL and the actual implementation of constraints with triggers and procedures in the DBMS, therefore assertions will be discussed further in Subsection 7.8.1. This section will further present the implementation of constraints for the landscape modelling system SALIX-2 (see Subsection 7.8.2), in DBMS, using triggers (see Subsection 7.8.3).

Once the system is extended with support for constraints, the user should be informed about the kinds of constraints available. This can be a simple list of all the

maintained constraints, or a more sophisticated attempt-alert approach in the interactive edit environment.

The apparent benefit of front-end implementation is direct interaction with the user. For example, if a user places a plantation object in the VRML scene, fast feedback of the validity of this placement can be realised if the constraints are also maintained in the visual environment (e.g. the VR component of the SALIX-2 system). However, the possibilities of changing the constraints within the VR-environment are limited, because currently the constraints are 'hard-coded' in the VR application code, as in many other edit/simulation environments. In future development environments it should be possible to automatically generate the part of the VR program application code that implements the constraints (as specified in UML/OCL).

Database implementation offers better management of constraints. If the constraints are stored in a database, they are stored in a central place, easily accessible and therefore easily adaptable. If in the VR/SALIX-2 example the application only connects to the database when saving or loading a plantation plan (such as with SALIX-2), there is not a connection during the interactive creation (editing) of the plantation plan. So the user only gets feedback when the plantation plan is saved, not when the plantation object is placed. An obvious and simple improvement is to automatically connect to the database after the user finishes a 'logical edit unit'. In this way immediate feedback is given, generated by the database, but presented in the edit environment.

Supporting integrity constraints in all subsystems is probably the optimal solution. That is, storing the constraints in a (model) repository on a central location and encapsulating this information in the different subsystems. Using this approach the feedback of the system will be significantly improved. However, if the constraints would be independently implemented more than once, this may lead to inconsistency between the subsystems. Therefore it is very important that the implementation of the constraints in the VR-environment is automatically derived and consistency is guaranteed.

7.8.1 Support for constraints in DBMS

In this subsection the support for constraints within the SQL92 is presented and compared to the actual functionality available in a number of DBMSs. According to the standard, three types of constraint are defined:

1. domain constraints, for example enumeration and range types;
2. general constraints (assertions) for any situation; and,
3. base table constraints, related to tables.

Domain constraints are relatively simple and will not be further discussed. The two other types are more interesting. With *general constraints*, also called *assertions*, and a rich set of spatial operators, many of the different types of constraints described in this chapter can be specified (according to the SQL 92 standard). The syntax of an assertion is:

```
CREATE ASSERTION <assertion_name> CHECK <constraint_body>
```

This syntax is quite simple and straightforward. When an attempt is made to commit the changes in a database, after a set of updates, inserts, and deletes, the assertion is checked and if the expression evaluates 'true' than the commit succeeds, otherwise it fails and the database remains in the old state. One could easily imagine automatic generation of these assertions from the UML/OCL invariants in ways similar to those by which database table definitions (DDL) can be derived from UML class diagrams.

To illustrate this a few examples will be given. First, an integrity constraint involving an aggregation 'the maximum height of the trees should be less than 10 (metres)':

```
create assertion size_is_ok check
  ((select max(height) from tree) < 10);
```

The same constraint can be formulated differently with the 'exists' construction in SQL, again specified as an assertion:

```
create assertion size_is_ok2 check
 (not exists (select * from tree where height >= 10))
```

Next, a different example showing a constraint involving a topological relationship 'there should be no tree standing in the water':

```
create assertion tree_not_in_water check
 (not exists (select * from tree, water
    where inside(tree.loc, water.polygon)));
```

It is clear from these examples that assertions are very powerful as any thematic, temporal, topological and geometric condition can be specified, between any number of tables. So, if assertion could be automatically derived from the UML/OCL models, this would conveniently implement constraints in the database. However, despite the fact the assertions are part of the SQL92 standard there is apparently no current DBMS (commercial or non-commercial) supporting their implementation (Oracle, DB2, Ingres, Informix, PostgreSQL, MySQL). The alternative might be *base table constraints* as in theory they are functionally equivalent to assertions. For example the previous constraint 'there should be no tree standing in the water' can be written as the following base table constraint:

```
create table tree (id integer, height integer, loc point,
  constraint tree_not_in_water check
    (not exists (select * from water
               where inside(loc, water.polygon))));
```

However, again there is disappointment as the mainstream DBMSs do not support base table constraints with subselects. That leaves the question 'What types of database table constraints are supported?' Below in more detail are the four types of database table constraints that are supported in Ingres (and this is quite representative for other DBMSs):

1. Unique constraint

```
create table ape(name char(10) unique not null, ....);
```

2. Referential constraint

```
create table mary(id integer,
      ape_name char(10) references ape(name))
```

3. Primary key constraint

```
create table ape2(name char(10) primary key, ....);
create table mary2(id integer,
      ape_name foreign key (name) references(ape2));
```

4. Check constraint (this is the only one with some 'semantic' load)

```
create table nut(balance integer
      check (balance > 0), spending integer);
create table nut2(balance integer, spending integer,
      constraint not_too_much check (spending < balance));
```

Though useful, these four types of constraint are not powerful enough to support the (spatial) constraints of the examples. One last option available for general constraint implementation in DBMSs is the use of triggers (and often in combination with procedures). Below is a constraint, now implemented as a trigger in Oracle (and similar functionality is available in other DBMSs) that checks whether the value of 'a_value' is not above the allowed maximum.

```
create trigger not_too_much
 after insert or update of a_value on a_table

  DECLARE
   total number;
  BEGIN
   select sum(a_value) into total from a_table;
   if (total >= 100) then
    raise_application_error (
     num => -20000,
     msg => 'Cannot add/update "a_value", sum too big');
   end if;
  END;
```

The advantage of this solution is that, although not pleasant to code, it really works. In practice CASE tools can generate some parts of the code; e.g. Oracle's CDM Ruleframe (see Subsection 7.8.3 for more details). With respect to the support of constraints in DBMS, as mentioned in Section 7.1, specific support for topology structures is improving (this can be compared to the built-in support of referential integrity constraints). As already mentioned some available solutions for managing topology constraints within the database are:

- Oracle 10g (2003) spatial includes some initial support for topological structures (DBMS checks topological consistency; e.g. is a loop closed, no crossing edges),
- LaserScan Radius topology is a (Oracle) DBMS solution (LaserScan, 2003, Louwsma et al., 2003) and
- also more 'middleware' types of solutions are available with support for topology constraints; for example the ESRI geo-database (ESRI, 2002).

7.8.2 Example assertions for the landscape design case

Though assertions are not directly supported in the available DBMSs, they are compact representations of integrity constraints and form convenient intermediate representations for generation from the UML/OCL model, for their final implementation within the database. A number of example constraints from the landscape design system, SALIX-2 will be presented as assertions. The tables of SALIX-2 that are used in the assertions are: prcv_treesrd_point (plantation objects of type trees and bushes) and prcv_gvkrd_poly (ground surface with description water, paving, soft_paving, grass, and bridge). The examples introduced in Section 7.2, Table 7.1. and illustrated in the UML class diagram of Figure 7.1., will be used. The first example is 'Bushes never lie inside water' (note the use of the Oracle 'sdo_relate' operator):

```
create assertion constraint_1 check (not exists (
select * from prcv_treesrd_point t, prcv_gvkrd_poly g
 where t.treetype in ('CorAve', 'CorMas', 'RosCan')
   AND g.descript = 'water'
   AND sdo_relate (g.geom., t.geom., 'mask=inside,
      querytype=window')='TRUE'))
```

The second example is a metric constraint specifying 'Trees always have to be positioned > 1 metre from paving' (again, note the use of the Oracle spatial operator 'sdo_within_distance'):

```
create assertion constraint_3 check (not exists (
SELECT * FROM prcv_treesrd_point t, prcv_gvkrd_poly g
 WHERE t.treetype IN ('FraxExc', 'QueRob')
   AND g.descript IN ('paving', 'soft_paving')
```

```
        AND sdo_within_distance (g.geom., t.geom.,
        'distance=1') ='TRUE'))
```

These examples show how easily and naturally all kinds of constraints can be specified. The last example includes an aggregation function to specify the constraint 'There must always be at least three trees on a specified ground surface' (for this constraint the grass polygon with id 20 is used):

```
  create assertion constraint_4 check ( (
  SELECT count(t.treeid) FROM prcv_treesrd_point t,
prcv_gvkrd_poly g
   WHERE t.treetype IN ('FraxExc', 'QueRob')
     AND g.id=20
     AND sdo_relate(t.geom,g.geom,
        'mask=ANYINTERACT,querytype=window')='TRUE'
  ) >=3)
```

7.8.3 Triggers and procedures

The assertions are easy to specify, but as mentioned, not supported in mainstream DBMSs. Therefore triggers (and procedures) offer the only practical way to implement constraints. Triggers can be seen as small programs checking certain conditions and prompting alerts with respect to the conditions. Using triggers one can achieve the same effect as by defining assertions. A functionally correct, but not very efficient, way to implement the assertions would be to glue all Boolean expressions of the individual assertions into one large Boolean expression. During every commit to the database, this expression is checked (via a trigger and procedure) and if the result is false, the transaction fails. However, this is not an efficient implementation and in this subsection triggers and procedures will be used in a more 'customised' way. However, one has to develop the more specific code for the triggers (and the used procedures) oneself. The syntax of specifying a trigger is:

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
BEFORE | AFTER
INSERT OR UPDATE [<column(s)>] OR DELETE ON <table_name>
[FOR EACH ROW [WHEN (condition)]]
<trigger_body>
```

A partial example ('bush must not stand in the water') of an implemented constraint in a DBMS using triggers and procedures is given in (Louwsma, 2004). Within the front-end application a new object is created and an insert statement is sent to the DBMS. The statement is checked for integrity constraints and feedback is given through DBMS outputs. In order to avoid the 'low level' hand-coding of constraints (or business rules), Oracle provides a development tool called Custom Development Method (CDM) for automatically generating this code for the DBMS (Muller, 2002; Boyd, 2001). The CDM RuleFrame is the business rules implementation framework of CDM. The rules consist of three parts (Muller, 2002):

1. a function that indicates when the rule should be validated;
2. a function that performs the actual validation, when the previous function indicates the need; and,
3. a handling procedure, that manages the communication with the outside world.

CDM RuleFrame does not check business rules at the moment the user performs insert, update or delete statements. Rather, CDM RuleFrame stacks the rules that have to be enforced and checks them only at the moment of commit. Stacking the rules and the business rule enforcement is performed in the Transaction Management component.

7.9 Conclusion and future developments

7.9.1 Results obtained

Constraints have not attracted much attention in GIS, despite the need for them in dynamic GIS applications as illustrated in the four different cases where constraints were analysed. Each case had its own language specifying the constraints (natural text, XML format, SQL assertions). It was concluded that formalisation is needed and that (spatial) constraints must be specified in UML/OCL. This chapter also proposed a classification (taxonomy) of the constraint types relevant to a dynamic GIS environment based on: 1) number of involved objects/classes; 2) properties of objects and/or relationships between objects: metric, topological, temporal, thematic or mixed; 3) dimension (2D, 3D or 4D); 4) manner of expression ‘never may’ or ‘always must’; and, 5) nature of the constraint ‘physically impossible’ or ‘design objective’.

From the single UML/OCL model specification, implementation of different parts of the system should be automatically derived although current environments are not yet capable of supporting this. The implementation should be: 1) front-ended, allowing to generate direct feedback to the users during editing; 2) during data definition, thus making sure only valid data are stored and accepted; and 3) during the communication or data exchange in the case of loosely coupled clients and servers, thus making sure the client is aware of the constraints. Database assertions do seem quite close to the UML/OCL invariants and it is therefore rather disappointing that these assertions are not yet implemented in mainstream DBMSs. Currently, the implementation of the more complex constraint types, which includes nearly all spatial constraints, has to be realised by triggers and procedures.

7.9.2 Further investigations

In real interactive applications an end-user who is a non-programmer, such as a landscape designer, must be able to change, delete or make new constraints. Therefore finding an easy way to interactively specify constraints is required. This is closely related to modelling. For example looking to the UML class diagram with all relevant object classes and their (restricted) relationships.

The user should get visual ‘feedback’ during editing; e.g. red or green highlighted areas during insert. These highlights should be derived from constraints, the instance geometry in a DBMS and be based on spatial functionality such as buffers and overlays. An investigation should take place to see whether the highlighted areas can be created inside the DBMS (and presented as views) or using specific GIS software.

This research can be extended to 2½D or 3D referenced objects. The objects of interest are currently often limited to point objects, polyline objects and polygon objects, but should be extended to volume (polyhedron) objects. However, data types and operations for 2½D and 3D geometry in DBMS are not yet available, but will be indispensable for implementing 2½D and 3D constraints concerning 2½D or 3D objects.

One issue not yet mentioned in this chapter is that, with a number of user-specified constraints, conflicts may arise; therefore, a good mechanism has to be developed to check for this. When users can change the constraint definitions of an existing application, then a conflict check should take place. This is of concern for existing software. For example topology constraints can be implemented in the ESRI (2002) geo-database software, but there is no check as to whether these conflict with each other. This can result in unwanted system behaviour; e.g. ‘an infinite loop’ correcting one error, which results in another error. There will never be a correct situation satisfying both rules (integrity constraints) as they are conflicting.

Finally more research is needed for topics such as: the automatic translation of UML/OCL to XML schema (as used in the exchange); the automatic translation of UML/OCL models to the edit environment (as in the case of the topographic data maintenance application); the relationship between consistent data and operations (as illustrated in the Web feature service case) and extending the constraints to space-time/simulation. The ultimate goal would be to support an environment in which it is possible to delete, change and add new constraints and automatically rebuild new versions of the different subsystems, during edits, storage and data exchange. Good support for constraints in a dynamic GIS environment is essential for data quality control and decisions based on these data.

Acknowledgements

All persons and organisations involved in the case studies are acknowledged: VR landscape design (Jildou Louwsma, MSc student, Sisi Zlatanova, TUD supervisor and Ron van Lammeren, Wageningen University, research supervisor), Dutch Cadastre (Christiaan Lemmen and Peter Jansen), Topographic Service (Nico Bakker), Notary drafts parcel boundary (Thijs Brentjes, MSc student, Marian de Vries, TUD supervisor and Tom Vijlbrief, Cadastre supervisor). Elfriede Fendel is acknowledged for the careful proofreading of the first version of the manuscript. Many thanks to Roland Billen and Jane Drummond for their editorial support. The author expresses his gratitude to the research program 'Sustainable Urban Areas' at Delft University of Technology for making this publication possible.

References

Belussi, A., Negri, M. and Pelagatti, G. (2004) 'GeoUML: A Geographic conceptual model defined through specialization of the ISO TC211 standards', in *Proceedings 10th EC GI & GIS Workshop, ESDI State of the Art*, Warsaw, Poland, (23–25 Jun 2004).

Boyd, L. L. (2001) 'CDM RuleFrame – the business rule implementation framework that saves you work', *ODTUG 2001, Business Rules Symposium*, Oracle Corporation, iDevelopment Center of Excellence.

Brentjens, T. J. (2004) *OpenGIS Web Feature Services for Editing Cadastral Data; Analysis and Practical Experiences*, MSc Thesis Geodetic Engineering, Delft University of Technology.

Brentjens, T., de Vries, M., Quak, W., Vijlbrief, C. and van Oosterom, P. (2004) 'Evaluating the OpenGIS Web Feature Services protocol with the case study 'Distributed Cadastral Transactions'', in Fendel, E.M. and Rumor, M. (eds.) *Proceedings of the 24th Urban Data Management Symposium*, Chioggia (27–29 Oct 2004), ch. 25, pp. 11–22.

Clementini, E., Di Felice, P. and van Oosterom, P. (1993) 'A small set of formal topological relationships suitable for end-user interaction', in *SSD'93: the Third International Symposium on Large Spatial Databases*, Singapore, pp. 277–295, LNCS no. 692, Berlin: Springer-Verlag.

Cockcroft, S. (1997) 'A taxonomy of spatial data integrity constraints', *GeoInformatica*, 1,4, pp. 327–343.

Cockcroft, S. (2004) 'The design and implementation of a repository for the management of spatial data integrity constraints', *GeoInformatica*, vol. 8, no. 1, pp. 49–69.

Collins, F. C. and Smith, J. L. (1994) 'Taxonomy for error in GIS', in Congalton, R. G. (ed.) *Proceedings International Symposium on Spatial Accuracy in Natural Resource Databases "Unlocking the Puzzle"*, pp. 1–7, Williamsburg: American Society for Photogrammetry and Remote Sensing.

Date, C. J. and Darwen, H. (1997) *A Guide to the SQL Standard*, 4th edition, ch. 14, pp. 197–218, Boston, Massachusetts: Addison-Wesley.

Egenhofer, M. J. (1989) 'A formal definition of binary topological relationships', *Proceedings of 3rd International Conference on Foundation of Data Organisation and Algorithms*, Paris, pp. 457–472.

ESRI (2002) *Working with the Geodatabase: Powerful Multi-User Editing and Sophisticated Data Integrity*, ESRI.

Heim, M. (1998) *Virtual Realism*, New York: Oxford University Press.

Hunter, G. J. (1996) 'Management issues in GIS: Accuracy and Data Quality', in Hunter, G. J. (ed.) *Proceedings of Conference on Managing Geographic Information Systems for Success*, Aurisa: Melbourne, Australia, pp. 95–101.

Kwon, Y-M., Ferrari, E. and Bertino, E. (1999) 'Modelling spatio-temporal constraints for multimedia objects', *Data & Knowledge Engineering*, vol. 30, pp. 217–238.

van Lammeren, R., Ogrin, D., Marusic, I. and Simanic, T. (2002) 'Virtual Reality in the landscape design process', *Proceedings International Conference on Landscape Planning in the Era of Globalisation*, pp. 158–165.

Laser-Scan (2003) *Technical Product Description - Topology Users Guide*, Cambridge, UK.

Louwsma, J. (2004) *Constraints in Geo-Information Models Applied to Geo-VR in Landscape Architecture*, MSc Thesis Geodetic Engineering, Delft: Delft University of Technology.

Louwsma, J., Tijssen, T. and van Oosterom, P. (2003) 'Topology under the microscope', *Geoconnexion*, vol. 2, no. 6, pp. 29–30.

Muller, S. (2002) *CDM RuleFrame Overview: 6 reasons to get framed!*, Oracle Corporation, iDevelopment Center of Excellence.

OGC (2002) Vretanos, P. (ed.) *Web Feature Service Implementation Specification*, version 1.0, Reference number: OGC 02–058, OpenGIS Consortium Inc.

OGC (2003) Cox, S., Daisey, P., Lake, R., Portele, C. and Whiteside, A. (eds.) *OpenGIS Geography Markup Language (GML) Implementation Specification*, version 3. Reference number: OGC 02–023r4, OpenGIS Consortium Inc.

OMG (2005a) Unified Modeling Language: Superstructure, version 2.0, formal/05-07-04, August 2005, [Online], Available: http://www.omg.org/cgi-bin/doc?formal/05-07-04.

OMG (2005b) Unified Modeling Language - Object Constraint Language (UML-OCL) 2.0 Specification Version 2.0, ptc/2005-06-06 June 2005, [Online], Available: http://www.omg.org/docs/ptc/05-06-06.pdf.

van Oosterom, P. (1997). 'Maintaining consistent topology including historical data in a large spatial database', *Proceedings of Auto-Carto 13*, Chrisman, N. (ed.), Bethesda: ACSM & ASPRS, pp. 327–336.

van Oosterom, P. and Lemmen, C. (2001) 'Spatial data-management on a very large cadastral database', *Computers, Environment and Urban Systems*, vol. 25, no. 4–5, pp. 509–528.

van Oosterom, P., Tijssen, T. and Penninga, F. (2005) 'Topology storage and use in the context of consistent data nanagement', *GISt report no. 33*, Delft, [Sep 2005].

Oracle (2003) *Oracle Spatial Topology and Network Data Models, 10g Release 1 (10.1)*, Author: Oracle.

Papadias, D. and Theodoridis, Y. (1997) 'Spatial relations, minimum bounding rectangles and spatial data structures', *International Journal of GIS*, vol. 11, no. 2, pp. 111–138.

Peuquet, D. J. (1995) 'It's about time: a conceptual framework for the representation of temporal dynamics in GIS', *Annals of the Association of American Geographers*, vol. 84, no. 3, pp. 441-461.

XSD (2004) XML Schema, [Online], Available: http://www.w3.org/XML/Schema, [14 Jul 2005].

Appendix 1. Cadastral constraint violation queries (CCVQ) in SQL

```
/* CCVQ1 */
SELECT object_id, numpoints(shape),
  anypoint(shape, 1), anypoint(shape, 2), anypoint(shape, 3)
FROM xfio_boundary
WHERE numpoints(shape)=3 and interp_cd=3 and tmax = 0 and
  ogroup = 6 and (anypoint(shape, 1) =  anypoint(shape, 3));
/* CCVQ2 */
SELECT object_id FROM lki_parcel
WHERE inside(location, geo_bbox) != 1;
/* CCVQ3 */
SELECT l_obj_id FROM lki_boundary
WHERE l_obj_id not in (select object_id from lki_parcel);
/* CCVQ4 */
SELECT object_id, fl_line_id FROM lki_boundary
   WHERE abs(fl_line_id) not in (select object_id from lki_boundary);

/* CCVQ5 */
SELECT object_id, line_id1 FROM lki_parcel
WHERE abs(line_id1) not in (select object_id from lki_boundary);
/* CCVQ6 */
SELECT s.object_id, s.fl_line_id
FROM xfio_boundary s, xfio_boundary r
WHERE s.fl_line_id > 0 and s.fl_line_id=r.object_id and
  s.tmax=0 and s.ogroup=6 and r.tmax=0 and r.ogroup=6 and
  s.r_obj_id <> r.l_obj_id;

/* CCVQ7 */
SELECT s.object_id, s.fl_line_id
FROM xfio_boundary s, xfio_boundary r
```

```
WHERE s.fl_line_id > 0 and s.fl_line_id=r.object_id and
  s.tmax=0 and s.ogroup=6 and r.tmax=0 and r.ogroup=6 and
  (anypoint(s.shape, 1) <>  anypoint(r.shape, 1));

/* CCVQ8 */
SELECT s.object_id, s.line_id1
FROM xfio_parcel s, xfio_boundary r
WHERE s.line_id1 > 0 and s.line_id1=r.object_id and
  s.tmax=0 and s.ogroup=46 and r.tmax=0 and r.ogroup=6 and
  (s.object_id <>  r.r_obj_id);

/* CCVQ9 */
SELECT count(*),municip FROM mo_object
WHERE pp_i_ltr='G' and x_akr_objectnummer not in
  (select x_akr_objectnummer from lki_parcel)
GROUP BY municip;
```

Appendix 2. Topographic model constraints in XML

```
TMCX1:
  <Domein>
        <Naam>dWaterBreedte</Naam>
        <Registreren>J</Registreren>
        <Beschrijving>Breedte voor Waterdeel</Beschrijving>
        <Type>Range</Type>
        <DataType>int</DataType>
        <CodedValueData/>
        <RangeData>
               <Minimum>1</Minimum>
               <Maximum>500</Maximum>
               <Default>6</Default>
        </RangeData>
        <SplitRegel>Duplicate</SplitRegel>
        <MergeRegel>DefaultValue</MergeRegel>
  </Domein>

TMCX2:
  <Domein>
        <Naam>dSpoorTypering</Naam>
        <Registreren>J</Registreren>
        <Beschrijving>Typeringen voor Spoorbaandeel</Beschrijving>
        <Type>CodedValue</Type>
        <DataType>int</DataType>
        <CodedValueData>
               <Code>44</Code>
               <Value>verbinding</Value>
               <Default>J</Default>
        </CodedValueData>
        <CodedValueData>
               <Code>45</Code>
               <Value>kruising</Value>
               <Default>N</Default>
        </CodedValueData>
        <RangeData>
               <Minimum/>
```

```
                <Maximum/>
                <Default/>
        </RangeData>
        <SplitRegel>Duplicate</SplitRegel>
        <MergeRegel>DefaultValue</MergeRegel>
  </Domein>

TMCX3:
  <AttribuutRegel>
        <Nummer>att007a</Nummer>
        <VervolgNummer/>
        <Categorie/>
        <Beschrijving>Als Naam_Aweg is ingevuld,
                dan WegType moet 'autosnelweg' bevatten</Beschrijving>

        <FoutMelding>WegType bevat niet autosnelweg</FoutMelding>
        <TriggerNiveau>1</TriggerNiveau>
        <VervolgOperator/>
        <FeatureKlasse>EDT_WEG_VLAK</FeatureKlasse>
        <AlsAttribuut>NAAM_AWEG</AlsAttribuut>
        <AlsOperator>!=</AlsOperator>
        <AlsWaarde>""</AlsWaarde>
        <DanAttribuut>WEGTYPE</DanAttribuut>
        <DanOperator>MVCONTAINS</DanOperator>
        <DanWaarde>|autosnelweg|</DanWaarde>
  </AttribuutRegel>
TMCX4:
  <AttribuutRegel>
        <Nummer>top04</Nummer>
        <VervolgNummer/>
        <Categorie/>
        <Beschrijving>Indien Wegvlak overlapt met Wegvlak
                dan moet HOOGTENIVEAU verschillend zijn</Beschrijving>

        <FoutMelding>Wegvlak overlapt Wegvlak</FoutMelding>
        <TriggerNiveau>1</TriggerNiveau>
        <VervolgOperator/>
        <FeatureKlasse>EDT_WEG_VLAK</FeatureKlasse>
        <AlsAttribuut>OBJECTID</AlsAttribuut>
        <AlsOperator>AREAOVERLAPAREA</AlsOperator>
        <AlsWaarde>EDT_WEG_VLAK</AlsWaarde>
        <DanAttribuut>HOOGTENIVEAU</DanAttribuut>
        <DanOperator>!=</DanOperator>
        <DanWaarde>FEATURE2.HOOGTENIVEAU</DanWaarde>
  </AttribuutRegel>

TMCX5:
  <AttribuutRegel>
        <Nummer>brn01</Nummer>
        <VervolgNummer/>
        <Categorie/>
        <Beschrijving>Iedere feature moet een Bron
hebben</Beschrijving>
        <FoutMelding>Geen Bron</FoutMelding>
        <TriggerNiveau>1</TriggerNiveau>
        <VervolgOperator/>
        <FeatureKlasse>EDT_WEG_VLAK</FeatureKlasse>
        <AlsAttribuut>OBJECTID</AlsAttribuut>
        <AlsOperator>&gt;=</AlsOperator>
```

```
      <AlsWaarde>0</AlsWaarde>
      <DanAttribuut>OBJECTID</DanAttribuut>
      <DanOperator>BRONCOUNT&gt;</DanOperator>
      <DanWaarde>0</DanWaarde>
   </AttribuutRegel>
```