A Methodology for Modelling of 3D Spatial Constraints

Daniel Xu, Peter van Oosterom and Sisi Zlatanova

Abstract In this work we demonstrate a new methodology to conceptualise and implement geo-constraints in 3D, which has not been widely explored yet. This is done in four stages from natural language to implementation, in which geometric primitives and Object Constraint Language (OCL) play a crucial role to formulate the constraints. A database including various 3D topographic objects (e.g. buildings, trees, roads, grass, water-bodies and terrains) from CityGML (no constraints yet) is used as a case study to apply the developed methodology. In this research, a first attempt to formulate 3D geo-constraints in OCL is made. Unified Modelling Language (UML) class diagram has been extended with graphical symbols for indicating constraints between classes (in addition to the additional compartment within a class for a class constraint). These constraint expressions can be tested and translated to other models/implementations when the OCL standard is extended with spatial types and operations. During this research, new types of constraints are defined as follows: general-level constraints (applicable to all object sub-classes), parameterised constraints (containing numeric values, e.g. maximum distance), constraints allowing exceptional instances (to resolve cases that have not been defined) and constraints relating to multi-scale representations (to check the consistency between two levels of detail which model the same object). Additionally common sense rules to detect conflicting constraints are specified as well.

D. Xu
Accenture Technology Solutions, Amsterdam, The Netherlands
e-mail: xusifeng@gmail.com

P. van Oosterom (✉)
Faculty of Architecture and the Built Environment, Section GIS Technology, Department OTB,
TU Delft, Delft, The Netherlands
e-mail: p.j.m.vanoosterom@tudelft.nl

S. Zlatanova
Faculty of Architecture and Built Environment, Department of Urbanism,
TU Delft, Delft, The Netherlands
e-mail: S.Zlatanova@tudelft.nl

A. Abdul-Rahman (ed.), *Advances in 3D Geoinformation*,
Lecture Notes in Geoinformation and Cartography,
DOI 10.1007/978-3-319-25691-7_6

1 Introduction

Nowadays the field of geo-information is undergoing major changes, and the transition from 2D to 3D is having a major influence on these changes. A significant amount of 3D datasets are stored in spatially enabled databases. Experts are aware that new quality control mechanisms need to be built into the database systems in order to secure and guarantee data integrity.

Constraints set up rules that must never be violated and thus are used to maintain the data integrity. Many research project have investigated implementation of spatial constraint but still at a theoretical level. Few implementations at specific GIS application domain have been carried out but most of them are in 2D or in 2.5D (Louwsma 2004). But 3D city/urban models are becoming increasingly popular and the need for 3D data models is still growing. 2D models have already created such a big margin for errors, let alone adding one more dimension. An extension of current constraint theory from 2D to 3D would be a great help to stabilise the data.

A generic methodology in modelling 3D constraints is developed in this research. At first, constraints are designed and expressed using natural language. Then objects in the sentences are abstracted by geometric primitives, and their interrelationship by topological relationships. By doing so, spatial constraints become more specific and clearer to the others. Following the well-defined spatial types and operations as proposed in the ISO19107 standard and using various tools, an attempt is made to formalise these constraints using OCL. Finally, the constraints are translated to executable code, e.g. Procedural Language/Structured Query Language (PL/SQL), and with a small modification realised in the database by trigger mechanisms.

A database including various 3D topographic objects and their attributes is selected as a case study to test this approach. Objects such as buildings, trees, sensors (stationary and mobile), land use (e.g. grass fields, water bodies, roads) and terrain are modeled and stored in the Oracle Spatial (11g) database (Geomatics 2010). This database plays a key role in Campus Spatial Data Infrastructure to store, retrieve and exchange the information of in-campus objects for climate study, e.g. simulation and analysis.

This paper is organised as follows: Sect. 1 is the introduction; Sect. 2 reviews the related work presented in papers; Sect. 3 presents details of modelling 3D geo-constraints in the four stages; Sect. 4 shows how this methodology can be applied to a 3D topographic model and pseudo 3D OCL expressions; Sect. 5 tells new insights of constraint nature and classification methods/classes; Sect. 6 gives conclusion and recommendations for future work.

2 Related Work

In order to define the constraints, the classification of typical constraints has been studied. The main types of traditional (explicit) constraints include domain constraints, key and relationship structural constraints, and general semantic integrity

constraints. These classical types have been then extended by (Cockcroft 1997) with new spatial classifications: topological, semantic and user-defined constraints. This classification has been again elaborated by (Louwsma et al. 2006) with different types of relationship: direction, topology, metric, temporal, quantity, thematic, topological and mixed, and was also proven by (van Oosterom 2006).

A commonly adopted method of modelling geo-constraints is OCL, as suggested by research projects (Casanova et al. 2000; van Oosterom 2006; Duboisset et al. 2005; Louwsma et al. 2006; Pinet et al. 2007), ISO standard (ISO 2008) and data specifications of INSPIRE (INSPIRE 2010a, b). OCL is a formal language used to describe the constraints applying to objects, and is part of UML which is a preferred concept modelling schema. An advantage is that it can be translated by available tools, e.g. Dresden OCL2 toolkit, into executable code such as Java or SQL. And with small modifications database triggers can be formulated from SQL code.

Beside OCL, there are also other ways to express constraints, such as the constraint decision table by (Wachowicz et al. 2008), meta data repository by (Cockcroft 2004), ontology by (Mäs et al. 2005), GML/XML by (Reeves et al. 2006). Nevertheless, unlike the rich results of theoretical study, literature gives much less examples of implementing constraints in the GIS application domain. Some of them are found in a landscape design system (Louwsma et al. 2006), field data capture system (Wachowicz et al. 2008), an agricultural information system (Pinet et al. 2004), a cadastral data maintenance system (TOP10NL) (van Oosterom 2006), a land administration system (Hespanha et al. 2008), and a traffic flow control system (Reeves et al. 2006). These examples only deal with 2D constraints.

A fast growth of 3D GIS models can be observed since the emergence of CityGML. It is an information model to represent 3D urban objects, which also considers multi-scale modelling. A single object is able to have four levels of details, each of which corresponds to a certain geometry. Although CityGML does not define constraints, the classes and relationships of objects in a city, including their spatial (geometric/topological), semantic and appearance characteristics, are modelled therein (Kolbe et al. 2009).

As the spatial model evolves from 2D to 3D, more and more database vendors support 3D data storage in their products. An example is Oracle. Since Spatial 11g, 3D object data types are supported, such as 3D (multi) points, (multi) lines, (multi) polygons and (multi) solids. 3D data geometry can be validated according to GML 3.1.1 and ISO 19107 specifications, e.g. geometry validation via SDO_Geom.Validate_Geometry_With_Context (Oracle 2010b). With the hierarchy of single objects and composite objects, it enables storing multiple Levels of Detail (LoDs) from City GML, which can model regions, cities/sites and architectural interiors. 3D queries concerning 3D visibility, volumetric analysis and spatial/semantic attributes are available in 11g (Ravada 2008) as well.

Despite the fact that 3D GIS is rapidly advancing and databases are able to implement geometric constraints, so far little research has been carried out on 3D constraints at a database level. Detailed research about implementing 3D constraints in databases is still highly demanding.

3 A Methodology for Geo-constraints Modelling

A methodology of modelling 3D geo-constraints is proposed here and can be used as a generic approach for all spatial-related constraints specifications in four stages:

1. Natural Language
2. Geometric/Topological Abstractions
3. UML/OCL Formulations
4. Model Driven Architecture (MDA)
    a. Database PL/SQL Code
    b. Exchange Format XML
    c. Graphic User Interface ArcGIS

Natural language is a simple way to specify a constraint statement relating to spatial objects. But it is subjective to the individuals and therefore a more objective specification is necessary. A logical next step is making drawings of the objects (mostly the 'nouns' in a sentence) in order to illustrate the shape of the objects. After that, the objects interactions (mostly the 'verbs') can be explained better by formal descriptions of topological relationships, e.g. Egenhofer 9 intersection matrices (9IM) (Egenhofer 1989). Constraint statements thus become more specific and clearer to the others and not subject to multiple verbal interpretations. In order to let machines understand the constraints and automate the model translation, a further specification should be made considering MDA. UML/OCL as a modelling aid/tool therefore is the choice at this stage. Under the support of various tools/software, the constraints implementation in the database (e.g. PL/SQL code), data exchange (e.g. XML schema), graphic user interface (e.g. ArcGIS) or any other domains can be automated. Here we focus on the constraints implementation in the database. With a small modification the generated code can be used in database triggers, which finally realises the implementation of constraints check. The detailed analysis of expressing and translating constraints will be described in the section below.

3.1 Constraints in Natural Language

Defining geographic constraints by natural language is a challenging task. Natural language is not formalised as machine language but subjective to individuals' interpretations of real-world phenomena. Not every geographic rule can be expressed as clearly and precisely as non-geographic rules, e.g. 'a book from the library can only be lent to one person at a time', or 'the grade of a supervisor must be higher than his supervisees' grade'.

An example of geo-constraint is 'a road cannot cross a building'. The designer would mean to avoid a clash between a car and a building. But the statement might be confusing if one is confronted with the data model which exactly depicts a real world scenario as in Fig. 1. A person may start wondering if this constraint is satisfied

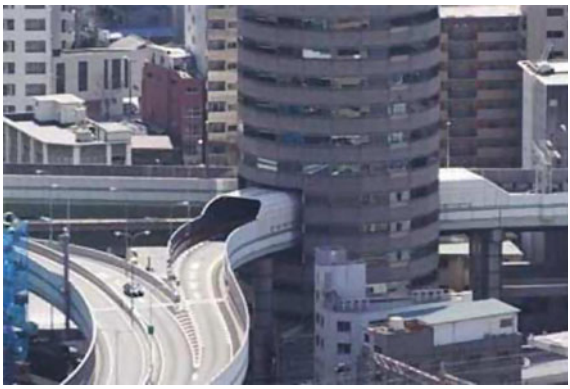

Fig. 1 An innovative architectural design where a building appears to be crossed by a road

by the model or not. Therefore it is difficult to avoid confusion when formulating constraints merely by natural language.

3.2 Geometric and Topological Abstractions

In the spatial model, the real world objects can be described by clearly-defined geometric primitives (e.g. solid, surface, line, point). Then the exact shapes of an object become clear, and the geometric primitives can also be specified by the components: interior, boundary and exterior. Based upon these primitives, the spatial relationships between objects can be formulated by 9IM and its extensions, such as calculus-based method from (Clementini et al. 1993) and 3D topological relationships from (Zlatanova 2000).

The example 'a road cannot cross a building' given in Sect. 3.1 can be specified using this abstraction. Here, three things in the text need to be clearly defined, 'what is a building?', 'what does cross mean?', and 'what is a road?'.

- A building: is a structure consisting of a set of surfaces that divide the whole space into buildings interior and exterior;
- A road: is a path that allows vehicles to travel along it at a certain speed;

Therefore we can model the building using *solid* geometry and the road *surface* geometry. And the term 'cross' can be replaced by 9IM 'intersect'.

The situation the constraint intends to forbid can be rephrased as:

'A surface must not intersect a solid'.

By using 9IM, the real world scenario in Fig. 1, which would be modelled as the one in Fig. 3, can be distinguished from the forbidden situation as illustrated in Fig. 2.

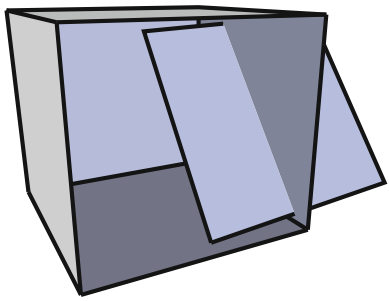

Fig. 2 An illustration of 'a surface intersects a solid'

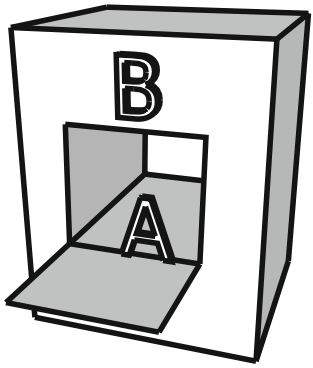

Fig. 3 Geometric model for Fig. 1. A is a road surface 'through' the hole in solid building B. In this case the constraint 'road A cannot cross building B' is not violated

3.3 UML/OCL Formulations

The UML diagram together with OCL notation has been found to be a suitable tool to express the designed constraints. An advantage of UML/OCL, comparing it to text-only alternatives, is that it unveils the relationships between objects visibly in diagrams. The OCL notations give formal (well defined and unambiguous) meanings that can be understood by both humans and machines. Another advantage is that it allows the designer to represent constraints on both geographic and alphanumerical data. And these two types of object can be used in one OCL constraint. For example, to indicate that a topological relation between geographical objects depends on the value of alphanumerical attributes.

UML is a common modelling tool and has a standard way to describe object classes, attributes and their interrelationships. Various tools support automatic generation of, e.g. SQL script, from UML class diagrams so that an implementation of the model can be created in the database by a few clicks. If OCL constraints can also be integrated to this code generation, the constraints can be integrated to database model and function through the whole lifespan of the database.

A challenge of specifying a relationship constraint is that currently there is no a rigorous mechanism in UML2.2 to indicate a constraint. When two object classes are related in the standard way (i.e. by association, specialisation, composition or aggregation), the constraint can be attached to the association. But when two object classes are not explicitly connected with an association link, the method to mention constraints about their relationships could lead to discussion.

For example, in general a road class and a building class may not have an explicit association. However, when some spatial constraint such as minimum distance between them is to be specified, a constraint association between the road class and the building class needs to be considered. Inspired by (Louwsma 2004), the normal association plus colour 'red' is proposed in this research as a new type of association link in UML class diagram (see examples in Figs. 6, 8 and 9 in Sect. 4).

A consequence of 'borrowing' association is the consideration of multiplicity. Because the constraint association will deal with class level rather than instance level, multiplicity which applies to instance level will not be considered.

Another difficulty which is somewhat related to the lack of constraint association in UML is that OCL itself does not support constraint expressions that have multiple classes involved. In normal OCL formula, if somebody wants to express a constraint related to a different class than the context class, he has to use the name of association end role to navigate from one class to the other. For example, to specify the no-intersect rule between a building instance and a road instance, the class 'Road' must be available in the context 'Building'. In other words, the class 'Road' should have a property that is of 'Building' type, or 'Building' class should have a property that is of 'Road' type. But in this example of no-intersect between a road and a building, neither 'Building' nor 'Road' has a property to type (typify) each other. So an expression can be:

```
context Building
inv BldRoadNoIntersect:
  Road.allInstances()->forAll(r | intersect(r.geometry, geometry)
= false)
```

Here, an extension of multiple classes in OCL is necessary to enable a 'navigation' from the context class `Building` ('self') to another class `Road` without an explicit association link. This extension can also be accompanied with our proposed constraint association in UML through which multiple class navigation is possible. More example expressions using multiple classes can be found in the pseudo OCL expressions in Sect. 4.

A workaround can be to set the context in a super-class which is shared by all the classes in the diagram. But this requires an ultimate super-class which can be navigated to by all classes, which is not always desirable. Therefore, it is better to extend OCL with multiple class expression.

3.4 Code Generation—Focus on Database PL/SQL Code

The Model Driven Architecture principle, being supported by Object Management Group (OMG), provides a framework to define how models in one domain-specific language (e.g. UML, OCL) can be translated to models in the other languages (Kleppe et al. 2003). In this paper, we focus on the implementation of constraints in an object-relational database. Some software/tools that are found useful to translate UML/OCL formulations into PL/SQL code are studied in this research. Their strengths and weaknesses in terms of OCL translation are given below.

- Enterprise Architect: EA is a UML drawing software used by ISO TC211 and INSPIRE for defining and visualising their data models and standards (Truyen 2005). A UML model can be automatically translated into Data Definition Language (DDL) and SQL script and thereby creates a corresponding database schema. Its OCL exporting function was used in the research project presented in (Hespanha et al. 2008). But except the referential key (Primary Key/Foreign Key) constraints and check constraints, the standard EA templates do not support the translation of other constraints such as OCL constraints to DDL or other implementation model. In our research, the standard syntax checking of EA appears to be weak and unreliable. Statements which are obviously incorrect in syntax can still be validated successfully. And spatial data types and operations are not supported.
- Dresden OCL2SQL: It's a part of Dresden OCL2 toolkit which provides a set of tools to parse and evaluate OCL constraints on various models, i.e. UML. Basically, it can transform a constraint to a SQL query that finds out all records that violate this rule this SQL code is integrated into triggers. The input for code generator includes a UML diagram of the appointed model and the associated OCL notations. Given a UML model together with correctly parsed constraints, within several clicks an executable SQL script can be generated.

For spatial constraints, the 3D geometries standardised in ISO19107 and 9IM topological names are not yet included in the OCL library. Articles from Pinet et al. (2007) and Werder (2009) show the possible extensions for 2D objects and 2D topological relationships. To achieve automatic SQL generation the extensions toward 3D, spatial types and operators can be added in a similar manner.

PL/SQL is Oracle's procedural extension language for SQL and its relational database. It combines the data manipulating power of SQL with the processing power of procedural language (Oracle 2010a). PL/SQL enables a trigger mechanism which is used to monitor the database and take a certain action when a condition occurs (Elmasri and Navathe 2003).

When a user modifies (create/insert/update/delete) certain datasets and then tries to commit the modification to the database, the trigger will be fired. Once it detects that a constraint is not satisfied in this commitment, it will give an error message to the front-ends and reject the transaction. By this means, a trigger is able to response to the data modification at run-time and guard the database integrity.

Given the trigger mechanism, if the OCL expressions are translated into SQL scripts, the spatial constraints check can be carried out by the spatial functions (e.g. distance(), buffer(), intersect()) supported by the database. In this sense, the power of data maintenance and spatial functions from database can be combined to have 3D geo-constraints integrated in database seamlessly.

However, the existing 3D functions in Oracle Spatial are relatively new and not extended. Many spatial and topological constraint checks can not be immediately implemented yet. The most useful function in Oracle Spatial database to calculate 3D topological relationships is `SDO_AnyInteract`. It is able to detect if two 3D objects are 'disjoint' or not. But it does not tell more details about what is happening in the 'non-disjoint' part. For example, two geometries which have 'touch' and 'intersect' do not make any difference to it. To be able to distinguish them in 3D, a new function named '3D_SurfaceRelate' is developed in this research.

4 Examples of Constraints on 3D Objects

The methodology (1. natural language, 2. geometry/topology, 3. UML/OCL, 4. implementation) can be applied to many 3D topographic models, such as city models. CityGML and more specifically 3DCityDB (Kolbe et al. 2009) was selected as a 3D topographic model in this project. Similar to other research project, such as (Emgård and Zlatanova 2008) which extends the standard CityGML model, we also modified the model to fit the needs of this research. A new class 'Sensors' was added to the model and represented with point geometry.

As was mentioned in related work (Sect. 2), not many OCL formulations found in the existing work deal with 3D geo-constraints, especially in the application domain. The necessary constraints regarding to city objects in 3D space are discovered and will be described in the following sections in natural language first. Then figures will be given to illustrate the geometric/topological abstraction of some of them. And a first attempt to formalise these constraints in UML/OCL—(*pseudo 3D Geo-OCL*)—will also be explained. As was mentioned in Sect. 3.3, the constraint association links use colour 'red' and the name of constraints is attached to the class diagram.

The well-defined 3D geometric primitives from ISO19107: 2003 standard (ISO 2008)—GM_Point, GM_Curve, GM_Surface, GM_Solid and the aggregational and compositional types of them—are used as spatial types in UML class diagrams.

Spatial operators from ISO19107 such as *distance()* and *intersect()*, as well as Oracle functions *inside()* and *validateGeometry()* are used in these formulations.

4.1 Generic Constraints

Some constraints are applicable to all classes. Instead of repeating them everywhere and to keep the UML diagram clean and brief, it is better to specify them on the

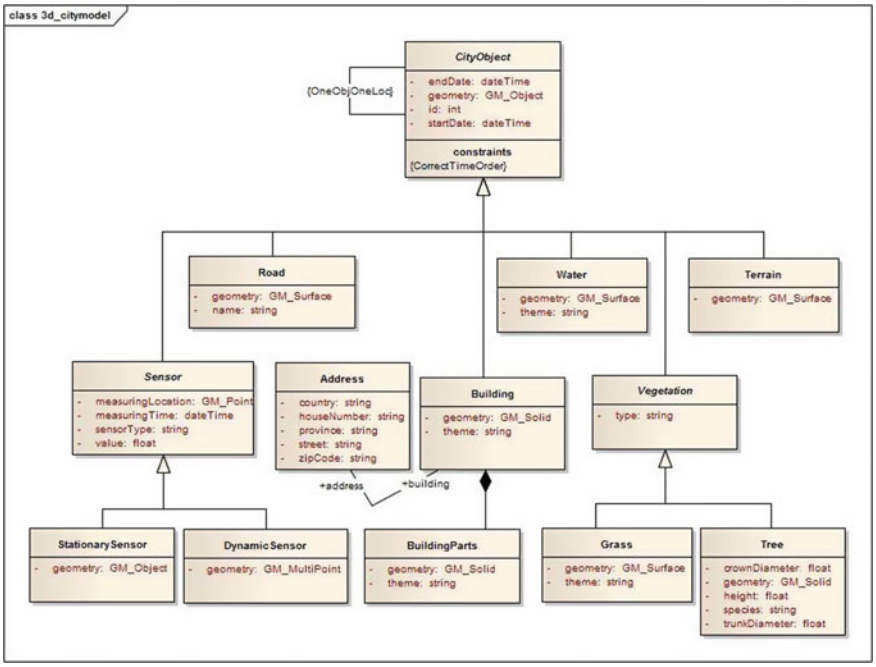

Fig. 4 UML model of constraints related to general city modelling (based on CityGML with extension on *sensor* class)

abstract/super-classes. With the inheritance, they can apply to all sub-classes (see `OneObjOneLoc` and `CorrectTimeOrder` in Fig. 4).

1. `OneObjOneLoc`: Given a local neighbourhood, there is only limited space to place an object. It is important to make sure the inserted/updated object do not clash into its neighbours (i.e. other buildings, trees) or stand on the wrong place (e.g. on the road, water).

A geometric/topological abstraction of this constraint can be seen in Fig. 5 where two solid objects intersect. This situation must be forbidden. Notice that this constraint applies to two or more instances under the same class. To distinguish this between-instances constraint from the constraint for single instance, e.g. alphanumerical constraint applying to attribute, a reflexive relation is used.

2. `CorrectTimeOrder`: The starting time and end time are general attributes for every object class. They can be used to review the changes of a certain area in time, e.g. land-use type for urbanisation monitor. It is important to keep these two time factors in a correct order—the starting time of an object must not occur after its end time.

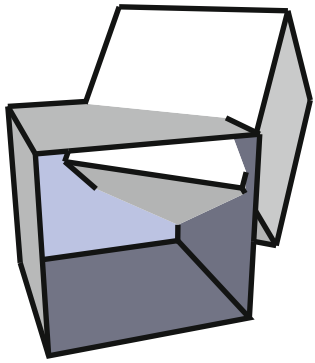

Fig. 5 Two solid objects intersect. A situation to be restricted by constraint `OneObjOneLoc`

```
context CityObject

inv OneObjOneLoc:
  CityObject.allInstances()->forAll(obj1, obj2 | obj1 <> obj2
 implies intersect(obj1.geometry, obj2.geometry) = false)

inv CorrectTimeOrder:
  startDate < endDate
```

4.2 Building Constraints

Building constraints—`ValidSolid`, `AddressMatchesRoadName`, `MinDistTreeBld`, `CorrectTimeOrderIn-Composition`, are attached to the UML class diagram in Fig. 6.

The extended symbol in UML class diagram—constraint association—is presented in the figure also.

3. `ValidSolid`: A building is treated as a composition of different parts. Each part can be modelled independently. This constraint checks if geometries (solids) of all building parts that belong to one building together form an (Oracle) valid solid. Oracle function *validateGeometry()* is used here which checks if the simple solids composing the whole object are topologically correct (adjacent).

An example of the geometric abstraction of this constraint is shown in Fig. 7. The three parts of a complex house must be adjacent and altogether form a valid solid.

4. `MinDistTreeBld`: This constraint checks if between one tree and one building the distance is larger than a minimum threshold. This is a parameterised constraint in which 'minDistBld2Tree' is an open parameter that can vary depending on different factors (see also Sect. 5.4).

5. `AddressMatchesRoadName`: The address of a building usually is related to its surrounding road. If a building in the model has a name that cannot be found in any name of roads in its local neighbourhood, this building must have been placed

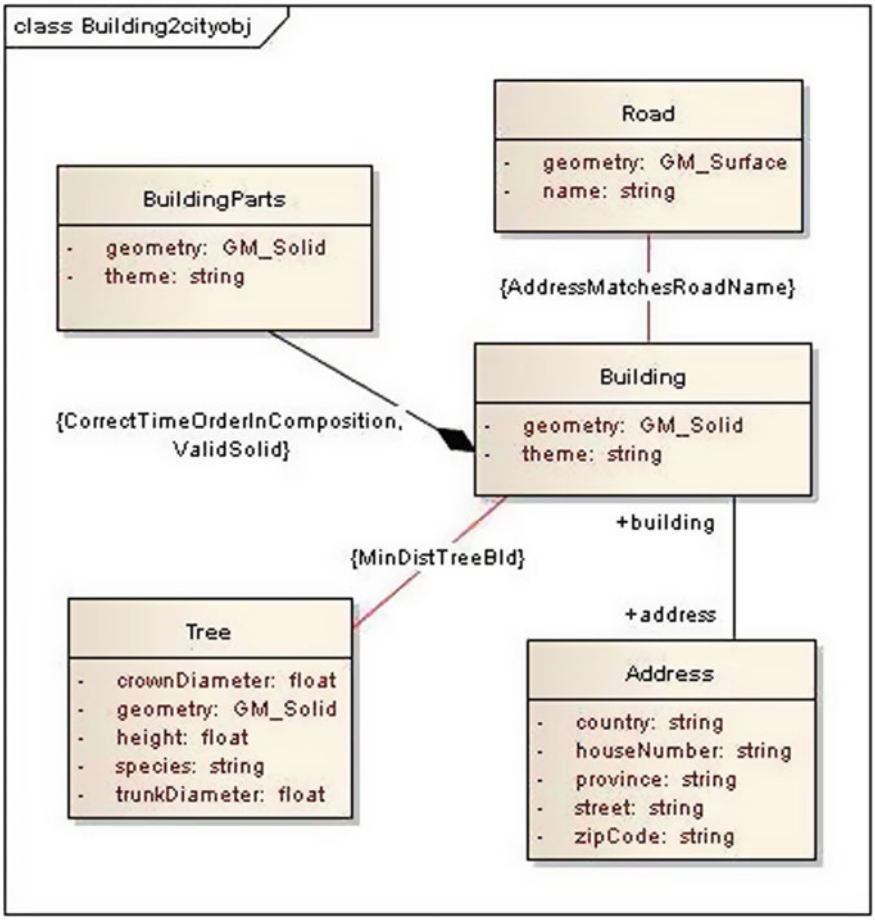

Fig. 6 UML model of constraints relevant to building object class

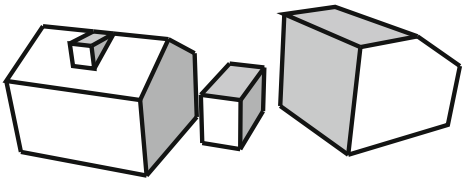

Fig. 7 The three parts of a house must altogether form a valid (e.g. Oracle definition) solid

in a wrong location. As a result, a relational constraint is specified to make sure a building is put in the right local area by comparing its address and the road name. This expression is also an example of using multiple classes in OCL.

6. `CorrectTimeOrderInComposition`: Since a building can be assembled by different parts and each part has its own starting time and end time, there is a temporal constraint taking place in the composition relation. The starting time of a part must be later than its assembly. And the end time of a part must occur before the end time of its assembly.

```
context Building
inv ValidSolid:
    buildingParts->validateGeometry(geometry) = true
inv MinDistTreeBld:
    Tree.allInstances()->forAll(t | distance(t.geometry, geometry) > minDistBld2Tree)
inv AddressMatchesRoadName:
   Road.allInstances()->one(r | distance(r.geometry,
 geometry) < maxDistBld2Road and address.street = r.name)
inv CorrectTimeOrderInComposition:
   buildingParts->forAll(b | b.startDate >= startDate) and
   buildingParts->forAll(b | b.endDate <= endDate)
```

4.3 Tree Constraints

Tree constraints—`MinDist2Tree`, `MinDist2Road`, `Aquatic_Close2Water`, `LimitedCrown/Height/TrunkSizes`—are attached to the UML class diagram in Fig. 8.

7. `MinDist2Tree/Road`: ensures minimum distances between two tree instances and between a tree instance and a road instance. These two are also parameterised constraints. Each distance rule can have a different min distance threshold (i.e. the min distance of tree-tree 'minDistTree2Tree' would be different from tree-road 'minDistTree2Road'). And the expression of tree-road constraint uses multiple classes.

8. `AquaticClose2Water`: Depending on the species, a tree object has a particular area to be planted in. Some aquatic plant can only grow close to or in the water. It is necessary to constrain the close distance of this species to water in the model. The expression below checks if an aquatic plant is close enough (comparing to a user-defined threshold) to water. This expression is parameterised and uses multiple classes.

9. `LimitedCrown/Height/TrunkSizes`: Every species of tree has a different limited size in terms of crown diameter, height and trunk diameter. It is unusual to see some bush plant growing into the size of an oak tree. Thereby, a rule to avoid unrealistic size of a tree is given. The species—maxCrownvalue/Height/Trunkvalue are constants for each species. But these parameters can differ from one species to another. Multiple-class expression is also used here.

```
context Tree
inv MinDist2Tree:
   Tree.allInstances()->forAll(t1, t2 | t1 <> t2 implies
```

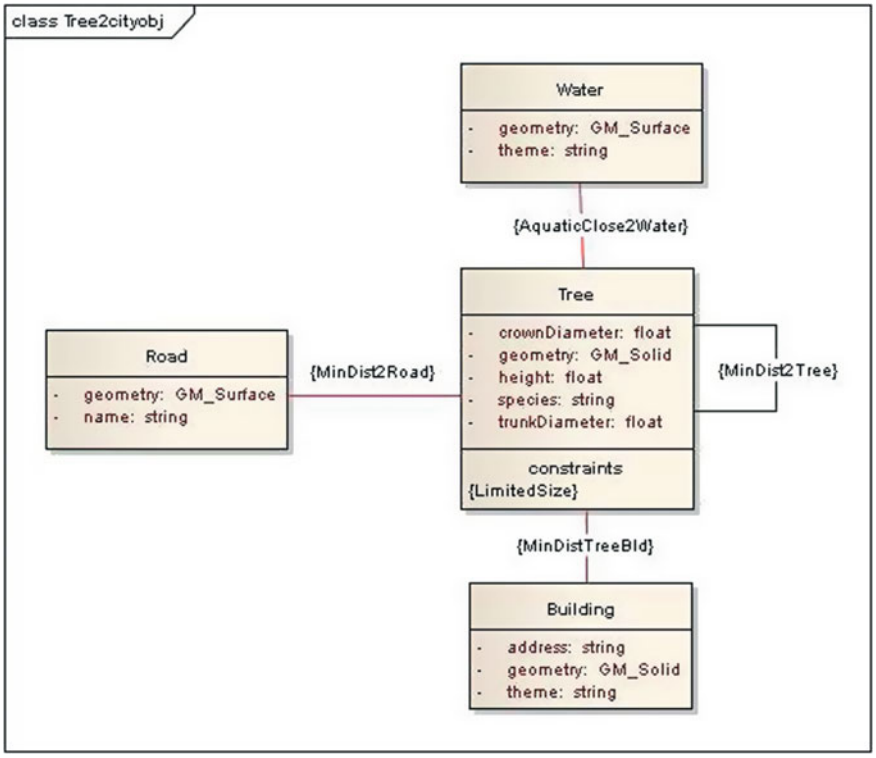

Fig. 8 UML model related to constraints in tree object class

```
 distance(t1.geometry, t2.geometry) > minDistTree2Tree)
inv MinDist2Road:
  Road.allInstances()->forAll(r | distance(r.geometry, geometry)
 > minDistTree2Road)
inv AquaticClose2Water:
  Water.allInstances()->exists(w | species = 'Aquatic' and
 distance(w.geometry, geometry) <= maxDistAquatic2Water)
inv LimitedCrownSize:
  crownDiameter<speciesMaxCrownValue
inv LimitedHeight:
  height < speciesMaxHeight
inv LimitedTrunkSize:
  trunkDiameter<speciesMaxTrunkValue
```

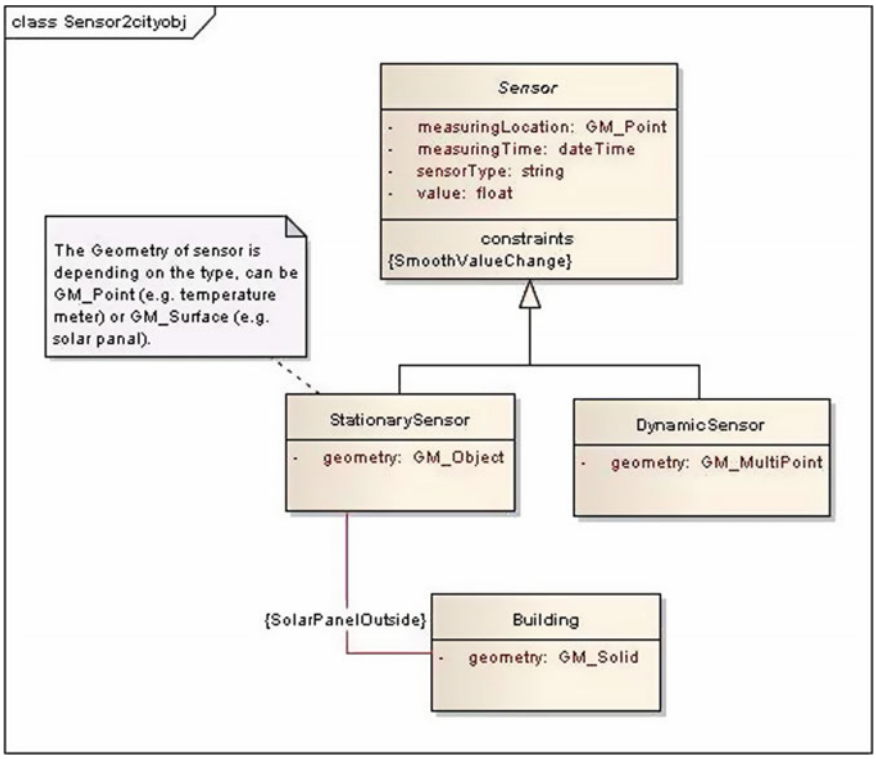

Fig. 9 UML model related to constraints in sensor object class

4.4 Sensor Constraints

The inheritance from generalisation is helpful in optimising the property constraints for sensors (static and mobile types). The necessity of having measurements of location and time for all sensors and smooth value changes is specified in the abstract type *Sensor* so that they can be automatically inherited by dynamic sensors and stationary sensors (see Fig. 9).

10. `SmoothValueChange`: Sensing data, which includes all different kinds of climate observations, is sensitive to erroneous value. One possible error in general is: two observations in a space of time have too big a difference. This is unusual for climate parameter(s), e.g. temperature, and thus needs to be marked as outliers.

The expression ensures that between two sensor instances from the same type the value change is within a threshold. The threshold parameter can be derived from change per time interval and vary with sensor type. Since this rule applies to every sensor object, it is added to the abstract class.

11. `SolarPanelOutside`: There are solar sensors (panel) equipped on some weather station and building roof. Usually the solar panels are installed on the roof,

or on the outside (wall) of building. Just shifting the panel object into building by 1 m can make its position become inside the building. This shift may not be obvious in terms of metric distance, but it definitely breaks the principle of using solar panel, which requires a direct reception of sunshine.

```
context Sensor
inv SmoothValueChange:
   sensorType = 'UserdefinedType' implies Sensor.allInstances()
->forAll(s1, s2 | s1 <> s2 implies abs((s1.value - s2.value)
 / (s1.measuringTime - s2.measuringTime)) <= userdefinedMaxChangeRate)
inv SolarPanelOutside:
   Building.allInstances()->forAll(b | sensorType = 'Solar Panel'
implies inside(b.geometry, geometry) = false)
```

As was discussed in Sect. 3.4, the current OCL standard does not support 3D spatial types and operations. Therefore, fully automatic PL/SQL code generation is not possible; the constraints need to be implemented in the database manually.

5 New Insights of Constraint Nature

During the research and the constraint design toward the case study, some general aspects regarding the nature and classification of constraint were discovered. Two ways of stating the same constraint, i.e. forced and restricted, were introduced from early literature by (Louwsma 2004). And in this research, a general rule at this point is specified to keep the statements brief and constraints easier to implement. New classes (types) of constraints are found, that is, constraints with/without exceptions, abstract/specific constraints, constraints with/without parameters, common sense constraints to detect conflicting constraints, and constraints related to consistency of multi-scale representations.

5.1 Forced and Restricted Specifications

There are basically two ways to formulate a constraint in natural language, forced (positive: must be/always have to) and restricted (negative, cannot/must not) (Louwsma 2004). Some constraints stated in one way would appear to be much more clear and efficient than stated in the other way. For example, the constraint 'grass colour always has to be green' has the same meaning as 'grass colour cannot have the colour black, white, gray, blue, red, yellow, purple'. The former is a forced way and is much briefer than the latter—the restricted constraint. As you may notice, what matters is the number of defined situations/values. 'Grass colour should be green' only defines one situation, whilst the other way defines many, potentially unlimited situations.

In the implementation stage, the difference does not only stay in length of text but also the performance of constraints. For example, constraint `OneObjOneLoc`

in Sect. 4.1 employs a restricting statement. According to that statement, an error will be given once the database detects relationship 'intersect' (9IM). The forced way in OCL-like manner can be:

```
inv OneObjOneLoc:
  CityObject.allInstances()->forAll(obj1, obj2 | obj1 <> obj2
implies disjoint(obj1.geometry, obj2.geometry) = true or
touch(obj1.geometry, obj2.geometry) = true)
```

which states the relationships can be 'disjoint' or 'touch' (and even more relationships that are allowed). If we would implement this statement, then at least two different topological relationships (instead of one—'intersect') have to be calculated before the database can tell if the constraint is satisfied.

In order to reduce the complexity of constraint specifications and implementations, a constraint design principle is that:

'All constraints should be compared using both statements. The statement that requires fewer values should be used'.

Of course, which way is more efficient may not be universal for all cases, but depends on the specific constraints and implementation.

5.2 *Constraints With/Without Exceptions*

According to (Servigne et al. 2000) and (Werder 2009), exceptions make a certain constraint more usable and realistic. In city environment, some conditions do not have to be strictly forbidden for the whole city. They may in general be unusual or irregular but in certain cases can nevertheless happen. These situations can be coped with by using exceptions.

Take the constraint expression `MinDistTreeBld` (from Fig. 8 in Sect. 4.2) as an example. Let us assume the 'minDistBld2Tree' is two meters. In most cases this statement is satisfied. But in some specific instances, the tree just leans on the wall or stands closer to a building than the allowed distance. It is good to reveal the common distance relation between a building and a tree and keep the distance, but it is difficult to give a clear-cut rule which describes all situations that have to be rejected. Instead of using 'must be' or 'cannot have' and the exact value, this example distance condition can be stated in natural language as:

'a tree and a building have at least two meters in between' (should constraint in class level) *or* 'instance pair tree-building is an exception' (exception in instance level) = true

So the principle can be summarised as:

An instance can be accepted (i.e. is true) if the 'should' constraint is satisfied or this instance is marked as an exception. In OCL-like format it can be expressed as:

```
Exception
   obj1 : OclAny
   obj2 : OclAny
context Building
inv minDistBld2Tree:
   Tree.allInstances()->forAll(t | distance(t.geometry, geometry)
> minDistBld2Tree
   or
   Exception.allInstances()-> one(e | e.obj1.isKindOf(
Building) and e.obj2.isKindOf(Tree) and e.obj1.id = id and
e.obj2.id = t.id))
```

5.3 Abstract/Specific Constraints

Some rules, such as `OneObjOneLoc` in Sect. 4.1, as well as `SmoothValue Change` in Sect. 4.4, are applicable to many sub-classes. Some other rules, such as `SolarPanelOutside` from Sect. 4.4, are only applicable to specific object sub-classes. The former type of rules can be generalised into a higher level and inherited by subclasses. This distinction can lead to a new classification of constraints in a generic context—abstract/specific constraints.

5.4 Constraints With/Without Parameters

Parameterised constraints which function upon exact values of objects and their properties can be distinguished from non-parameterised constraints. Examples as such are `MinDist2Tree, MinDist2Road` in Sect. 4.3. It is difficult to assign an exact value to the class and reject all the instances which do not satisfy it, whilst the non-parameterised rules can use one statement (such as `ValidSolid` from Sect. 4.2) and reject all violating instances.

5.5 Common Sense Rules

When the constraints are many, it may happen that some rules contradict, which means no solution can be found to satisfy them all. In practice, the result is that no instance of a certain class can be stored in the database (some tables will always be empty). An example is made up here: three parameterised rules regarding distances amongst tree, building and road are proposed as (see also illustration in Fig. 10):

- Tree-Building: tree must be placed within a distance of five meters from building.

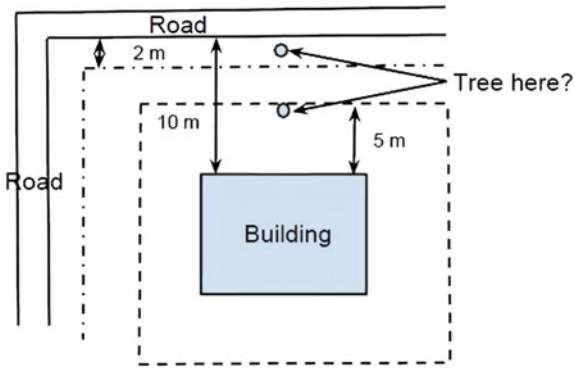

Fig. 10 A possible scenario of inserting a tree when tree-building-road distance constraints contradict

- Building-Road: there must always be at least 10 meters distance between a building and a road.
- Tree-Road: tree must be at a distance of less than two meters from a road.

As you can see, if buildings and roads are correctly stored, then it becomes impossible to have tree objects in the database. Because no solution can be found to satisfy both tree-building and tree-road constraints. Plainly, this outcome contradicts to the intention of having tree class in the database for it is created to allow storage of tree objects.

Explicit statement of this type of intention (let's call it common sense rules), which reflects human intuition or common sense, is subtle (easy to neglect) and yet important for constraints reasoning. They can be considered as rules of rules and may not directly apply to a specific object/attribute/relationship in the model. Therefore the common sense rules should be distinguished from the rest.

5.6 Multi-scale Consistency Rules

When a real world object or phenomenon is modelled with multiple scales, e.g. CityGML LoDs, it is necessary to have constraints that keep the geometry consistency between two different scales. For example, the same building represented in a finer scale and a coarser scale under the same reference system must not have coordinates that are too far apart (e.g. 100 m) in space. Neither should the volumes of a structure be too different in different scales. This type of constraints deals with relationships between the same objects in different scales, instead of relationships between different objects in the same scale. Therefore they should be distinguished from the same-scale-dealing constraints.

5.7 Indoor Consistency Rules

The best example of 3D constraints can be seen in 3D indoor modelling. Almost all 3D CityGML LOD4, IFC and IndoorGML maintain volumetric objects for rooms (CityGML), construction elements (IFC) and/or spaces (IFC, IndoorGML). The very common constraint is then room/spaces should not 'intersect' or must 'touch'. In addition, some of the objects are nested in each other. For example a furniture is inside a room, or a window is inside a wall. This can generate a large number of topological relationships. Interesting example is IndoorGML. It is a relatively new standard providing a framework for deriving a network for navigation. The framework is based on duality graph and requires complete subdivision of space (topographic, security, sensor, etc.). Intersections of and gaps between spaces (navigable or non-navigable) are not allowed. The only possible relationship is 'touch'. Despite the obvious need for constraints in 3D indoor modelling, to ensure good quality data, these constraints are not yet formalized and not yet part of today's standards (IFC, IndoorGML). A similar approach as presented in this paper should be conducted to obtain the constraints. This is part of future work.

6 Conclusion

This paper demonstrates a developed methodology to model and implement 3D geo-constraints, following four stages: *natural language, geometric/topological abstractions, UML/OCL formulations, PL/SQL code*. This methodology also addresses the third dimension which has not been discussed much in present papers (mostly 2D), and can be applied to a broad context of geo-constraints modelling in both 2D and 3D.

A database in application domain that stores 3D topographic objects commonly seen in the urban environment was used to apply the methodology. Some constraints regarding the objects in 3D space have been formulated in this work. Because currently automatic model translation from OCL to SQL is not available, so PL/SQL code was written by hand. The challenges of automatic translation lie on the support of spatial types and operations in OCL standard (see also Sect. 4), multiple class expression of OCL (see Sect. 3.3) and sufficiency of spatial functions in the database (see Sect. 3.4).

New observations about constraint nature have also been discussed (in Sect. 5). Two ways of specifying a constraint can have sharp difference in terms of statement briefness and performance. New classes and classification methods have been found and proposed for a better understanding of constraints.

During this research, standards have been studied and used. The experience gained in using them has led to the conclusion that some improvements could be made by the owning organisations.

- OCL: It would be highly productive to extend current OCL standards with 3D spatial types/operations and topological relationships (see Sect. 3.4).
  The extension with multiple class expression is also highly demanded (see Sect. 3.3) which can also be accompanied with the extension of the constraint association in UML.
- CityGML: It would be advisable to have geo-constraints incorporated into the UML model, such as multi-scale consistency constraints (see also Sect. 5.6), so that city modelling could be more concise and well-defined (see also Sect. 3.4). Also indoor constraints should be developed (as in CityGML LoD4 and the new IndoorGML) which covers both Indoor-Indoor consistency and Indoor-Outdoor consistency. Examples for Indoor-Indoor are that rooms, corridors and other building interiors should not overlap but connect and partition whole building. Examples for Indoor-Outdoor are that indoor elements should be inside the outdoor model (envelope). This at all stages in time, both old, existing and future (planned) buildings.
- UML: It would be recommended to create a symbol—constraint association—to connect two classes that do not have normal types of association, and also allow using multiple class in OCL (see Sect. 3.3). At the finishing of this paper, research from (Egenhofer 2011) and [citemas2008reasoning was found which add multiplicity to the relational constraints between two classes with e.g. for-all, for-some. Therefore, the multiplicity (at instance level) of the involved classes can be added to the proposed constraint association. In order to avoid its inconsistency with the OCL expression, it would be better if this multiplicity is generated by the UML editor, e.g. EA or Eclipse.

Suggestions regarding to future work for researchers who wish to extend the work carried out in this research are:

1. The pseudo 3D Geo-OCL expressions from Sect. 4 need to be tested in conjunction with the UML diagrams.
2. It would be useful to extend EA, or Dresden OCL2SQL or other OCL code generation tools to enable automatic model translation from OCL (esp. spatial constraints) to SQL (see also Sect. 3.4). This of course can also be provided by software developers.
3. Further study can be conducted into detecting contradicting (spatial and non-spatial) constraints, as explained in Sect. 5.5. It can be done by using OCL verification tools, e.g. HOL-OCL, UMLtoCSP (Chimiak-Opoka et al. 2010). Or through First Order Logic reasoners that are adapted/extended to understand geographic predicates and types.
4. Corresponding functions in database need to be developed, esp. 3D and solids related, to implement 3D spatial predicates from extended OCL. This extension could be a common effort dedicated by the manufacturer, developers and researchers, as was presented in Sect. 3.4.
5. Test the performance of the trigger that uses 3D geometric operators. Geometric operations are often time consuming and it is often impossible to use trigger in

each update. Furthermore, when the amount of geometric objects in one update is great, how the performance of trigger will be is an interesting issue to study.

References

Casanova, M., Wallet, T., & D'Hondt, M. (2000). Ensuring quality of geographic data with UML and OCL. In A. Evans, S. Kent and B. Selic (Eds.), *UML'00 Proceedings of the 3rd International Conference on The Unified Modeling Language: Advancing the Standard* (pp. 225–239). Heidelberg, Berlin: Springer.

Chimiak-Opoka, J., Demuth, B., Awenius, A., Chiorean, D., Gabel, S., Hamann, L., et al. (2010). OCL tools report based on the IDE4OCL feature model. In *Proceedings of the Workshop on OCL and Textual Modelling (OCL 2010)* (Vol. 36, p. 18).

Clementini, E., Di Felice, P., & van Oosterom, P. (1993). A small set of formal topological relationships suitable for end-user interaction. In *The 3rd International Symposium on Large Spatial Databases*.

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

Cockcroft, S. (2004). The design and implementation of a repository for the management of spatial data integrity constraints. GeoInformatica 8(1):49–69.

Duboisset, M., Pinet, F., Kang, M.-A., & Schneider, M. (2005). Precise modeling and verification of topological integrity constraints in spatial databases: From an expressive power study to code generation principles. In L. Delcambre, et al. (Eds.), *Conceptual Modeling—ER 2005* (pp. 465–482). Klagenfurt, Austria: Springer.

Egenhofer, M. (2011). Reasoning with complements. In *Advances in Conceptual Modeling. Recent Developments and New Directions* (pp. 261–270).

Egenhofer, M. J. (1989). A formal definition of binary topological relationships. LNCS 367:457–472.

Elmasri, R., & Navathe, S. B. (2003). *Fundamental of database systems* (4 ed.). Addison Wesley.

Emgård, L., & Zlatanova, S. (2008). Implementation alternatives for an integrated 3D information model. In *Advances in 3D Geoinformation Systems, LNGC* (pp. 313–329).

Geomatics, M. (2010). Measure the climate, model the city—acquiring and storing temporal and spatial data for climate research. Technical report. GM2100 Synthesis Project, 112 pp.

Hespanha, J., van Bennekom-Minnema, J., van Oosterom, P., & Lemmen, C. (2008). The model driven architecture approach applied to the land administration domain model version 1.1, with focus on constraints specified in the object constraint language. In *FIG Working Week* (p. 19).

INSPIRE. (2010a, April 26). D2.8.i.5 INSPIRE data specification on addresses guidelines.

INSPIRE. (2010b, April 26). D2.8.i.6 INSPIRE data specification on cadastral parcels guidelines.

ISO. (2008, September 17). 19107:2003. In *Geographic information—Spatial schema*. ISO.

Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA Explained: The model driven architecture: Practice and promise*. Addison-Wesley Professional.

Kolbe, P. D. T. H., König, G., Nagel, C., & Stadler, A. (2009). *3D-Geo-Database for CityGML*. Institute for Geodesy and Geoinformation Science, Technische Universität Berlin, 2.0.1 edition.

Louwsma, J. (2004). Constraints in geo-information models—applied to geo-VR in landscape architecture. Master's thesis, TU Delft. GIRS-2004-17, 104 pp.

Louwsma J, Zlatanova S, van Lammeren R, van Oosterom P (2006) Specifying and implementing constraints in giswith examples from a geo-virtual reality system. GeoInformatica 10:531–550.

Mäs, S., Wang, F., & Reinhardt, W. (2005). Using ontologies for integrity constraint definition. In *Proceedings of the 4th International Symposium on Spatial Data Quality'05*, Beijing, China.

Oracle. (2010a). *Oracle Database, PL/SQL Language Reference, 11g Release 2 (11.2)*. E17126-04 edition, 756 pp.

Oracle. (2010b). *Oracle Spatial Developer's Guide 11g Release 2 (11.2)*, 916 pp.
Pinet, F., Duboisset, M., & Soulignac, V. (2007). Using UML and OCL to maintain the consistency of spatial data in environmental information systems. *Environmental Modelling & Software*, *22*.
Pinet, F., Kang, M.-A., & Vigier, F. (2004). Spatial constraint modelling with a GIS extension of UML and OCL. In U. K. Wiil (Ed.), *Metainformatics* (pp. 160–178). Salzburg, Austria: Springer.
Ravada, S. (2008). *Oracle spatial 11g technical overview*. Presentation. Retrieved from, December 2010.
Reeves, T., Cornford, D., Konecny, M., Ellis, J. (2006). Modeling geometric rules in object based models: An XML/GML approach. Progress in Spatial Data Handling 3:133–148.
Servigne, S., Ubeda, T., Puricelli, A., Laurini, R. (2000). A methodology for spatial consistency improvement of geographic databases. GeoInformatica 4(1):7–34.
Truyen, F. (2005). Implementing model driven architecture using enterprise architect—MDA in practice. White paper practice, Sparx Systems. Version 1.1.
van Oosterom, P. (2006). Constraints in spatial data models, in a dynamic context, Chapter 7. In *Dynamic and Mobile GIS: Investigating Changes in Space and Time* (pp. 104–137). CRC Press.
Wachowicz, S., Wang, F., and Reinhardt, W. (2008). Extending geographic data modeling by adopting constraint decision table to specify spatial integrity constraints, Chapter 2. In *The European Information Society: Leading the Way with Geo-information. LNGC* (pp. 435–454). Heidelberg, Berlin: Springer.
Werder, S. (2009). Formalization of spatial constraints. In *12th AGILE International Conference on Geographic Information Science*. Germany: Leibniz Universität Hannover.
Zlatanova, S. (2000). On 3D topological relationships. In *DEXA '00 Proceedings of the 11th International Workshop on Database and Expert Systems Applications* (pp. 913–919). London, UK: Greenwich.