



**Dipartimento di Informatica
Università degli Studi di Verona**

**Rapporto di ricerca 96/2015
Research report**

June 2015

Spatial Integrity Constraints in 3D City Models: from Conceptual Definition to SQL Implementation

**Alberto Belussi
Sara Migliorini
Mauro Negri
Giuseppe Pelagatti**

Questo rapporto è disponibile su Web all'indirizzo:
This report is available on the web at the address:
<http://www.di.univr.it/report>

Abstract

Several different models have been defined in literature for the definition of 3D city models, from CityGML [13] to Inspire [7]. Such models include a geometrical representation of features together with a semantical classification of them. The semantical characterization of objects encapsulates important meaning and spatial relations which are defined only implicitly or through natural language, such as buildings shall be disjoint or in touch, or a window surface shall be contained in the building boundary.

The problem of ensuring the coherence between geometric and semantic information is well known in literature. Many attempts exist which try to extend the OCL language in order to represent spatial constraints for an UML model. However, this approach requires a deep knowledge of the OCL language and the implementation of ad-hoc procedures for the validation of constraints defined at conceptual level.

The aim of this paper is the development of a set of templates for expressing spatial 3D constraints between features which does not require any particular knowledge of a formal language. Moreover, the constraints instantiated from these templates can be automatically translated into validation procedures, without the need for ad-hoc implementations.

Keywords: keywords

1 Introduction

Many different models exist in literature for representing 3D spatial data and in particular 3D city models, such as CityGML [13] and Inspire Annex 3 Building [7]. Besides to the geometrical characterization of objects, which essentially refers to the ISO geometric types [10], such models provide a semantical characterization of objects which encapsulates important meanings and spatial relations, some examples of these semantical descriptions (in form of constraints) are reported in Sec. 1.1.

However, semantical constraints are not directly integrated in the conceptual model but are usually implicit or expressed in natural language. Therefore, ad hoc procedures have to be implemented in order to validate and ensure the satisfaction of constraints defined at conceptual level. This introduces a gap between the conceptual design of a spatial dataset and its implementation on GIS systems. Conversely, the ability to define spatial integrity constraints at conceptual level allows designers to abstract from implementation details and to apply one common constraint framework.

The Standard ISO TC211 19109 “Rules for application schema” recommends the use of the OCL [11] language for specifying spatial integrity constraints at conceptual level. Sec. 2 will discuss several approaches to automatically translate spatial-enhanced OCL constraints into validation procedures. However, the use of generic OCL constraints has several limitations as discussed in [14]. In particular, it introduces a great complexity both in the conceptual modeling, since a deep knowledge of the OCL language is required, and in the implementation phase, since complex ad-hoc procedures have to be implemented for treating all cases and it is more difficult to optimize such procedures.

This paper follows the approach proposed in [14] that allows the definition of spatial integrity constraints into conceptual models through the use of predefined OCL templates. Thanks to these templates the designer can specify spatial integrity constraints in a straightforward manner, without the need for a deep knowledge about the OCL language. Moreover, such constraints can be automatically translated into SQL queries or into procedures in other programming languages, in order to verify the correctness of spatial datasets. Of course, it is much easier to optimize a set of constraint templates, because their structure is known.

More specifically, this paper extends the work in [14] to the 3D space, in particular as regards to surfaces and volumes. It takes as reference the building model of CityGML (LoD3) and INSPIRE Annex 3 Building, but the approach can be easily adapted to other generic 3D spatial models. The contribution of the paper is twofold: (1) it provides a set of templates that

can be instantiated for representing 3D topological constraints at conceptual level with reference to the standard ISO geometric model (Sec. 3-4), and (2) it demonstrates how such constraint templates can be automatically translated into validation procedures using first a mathematical geometric formulation (Sec. 5.1) and then with reference to the SQL language (Sec. 5.2). In particular, such implementation refers to the 3D geometric functions and data types available in the PostGIS system.

The choice to implement the validation procedures in SQL results particularly useful in the common case where validation tests have to be performed on a huge amount of 3D spatial objects that are quite simple (i.e., in a city model, not in a building model). However, the proposed template mechanism and the abstract mathematical formulation proposed in Sec. 5.1 can be implemented using other programming languages or technologies.

1.1 Motivating Example

This section illustrates some examples of 3D topological constraints among buildings and its constituent parts which can be useful in the definition of a 3D city model. Some of these constraints are taken from CityGML but can be reasonably applied to any other model. In particular, some concepts and definitions of CityGML [13] LOD3 building model are used: for instance, the thematic classification of openings and boundary surfaces.

First of all, in a city model all buildings shall be disjoint or touch each other. Moreover, if a building consists of only one (homogeneous) part, it shall be represented by a unique solid element. Otherwise, if it is composed by several individual structures, it shall be modeled as a set of solid parts, such that all these parts touch each other to form a composite solid, see Fig. 1. With reference to CityGML, each building part must be related to exactly one building and touch it.

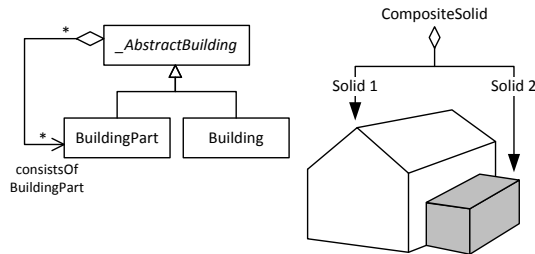


Figure 1: Example of building composed of two parts that touch each other.

With reference to the CityGML and/or INSPIRE building model, the outer facade of a building can also be differentiated semantically using a set

of surface types with a special function, like wall, roof, ground, and so on, see Fig. 2. Clearly, if a building is represented by both a solid and a set of boundary surfaces, these surfaces have to touch the boundary of the solid.

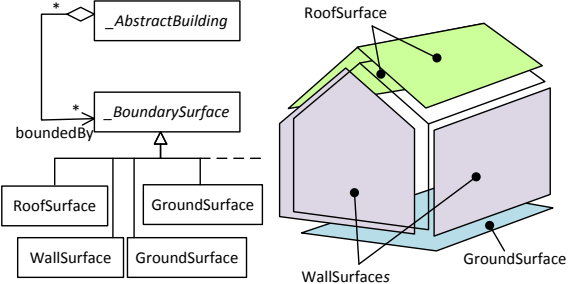


Figure 2: Example of building boundary surfaces.

The last but more common example regards the relation that exists between the outer shell of a building and its openings (i.e., doors and windows). Different representations can be adopted for modeling openings: in the simplest case, a building can be represented by a solid and its openings are surfaces that have to touch the solid. Conversely, in a more elaborated model, such as CityGML LOD3, openings are surfaces that can be related to one of the boundary surfaces representing the building outer shell. Moreover, since boundary surfaces have a precise semantic meaning, openings like doors and windows can be found only on roof and/or wall surfaces. In particular, if the geometric location of an opening topologically lies within a boundary surface component, then it must be represented as a hole within that surface: the opening surface must be embraced by a set of surfaces defining the building boundary. For instance, in Fig. 3 taken from [13], the window surface has to be embraced by some wall surfaces, the outer ceiling surface and the outer floor surface. Notice that the opening surface must not be contained in any building surface.

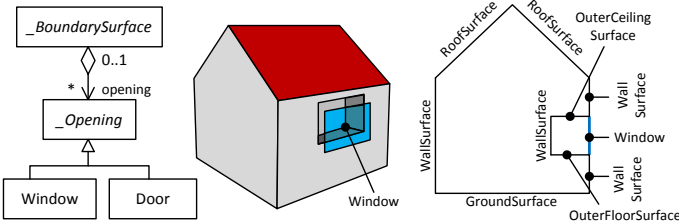


Figure 3: Example of building with an opening.

2 Related Work

In [15] the authors analyse the structure of semantic and spatial information of 3D city models as well as their correspondence. In particular, they distinguish the semantic model of CityGML from the geometry model. The semantic model consists of class definitions for the most important features within 3D city models, such as buildings, water bodies, transportation, vegetation, and so on; conversely, the geometry model is based on the Standard ISO 19107 “Spatial Schema” [10]. According to this separation, the term *coherence* describes consistent relationships between spatial and semantic entities. The author classifies six different situations depending on the degree of specification of both the semantic and geometric components. In particular, CityGML supports two cases: one in which only semantics is specified without geometry or vice-versa, and the other characterized by complex objects with structured geometry. Clearly, the last one requires the specification of more detailed consistency rules. The authors conclude that the coherence evaluation could be performed at data model level by explicitly marking some aggregation relations in the semantic model as spatial aggregations, but does not provide any tool for checking such constraints.

In [8] the author provides a set of axioms to achieve geometrical consistency of 3D models. This paper only deals with the geometric consistency with respect to the specified spatial data types. It provides the theoretical foundations for checking tools by defining an axiomatic characterization of 3D city models. The efficiency of the proposed method stems from its locality: in most cases, consistency checks can be safely be restricted to single components, which are defined topologically: a 3D city model is decomposed into simple components and the axioms can be combined in order to obtain constraints for aggregation of components. The geometrical-topological 3D city model is defined by a graph defined by a set of vertices, edges, faces and solids. Example of axioms are: each solid is bounded by a closed composite surface, each vertex and edge is incident to at least one face.

In [16] the authors define a set of axioms which regards both the geometric and semantic validation of a CityGML model. In particular, they define the concept of *valid geometry* using the definition of spatial data types provided in [13], and a set of axioms which check their compliance. Given a valid geometry, a set of *semantic rules* is defined which are mainly determined by the underlying data model. Such rules are defined in terms of OCL [11] statements which refer to the UML diagram defining the CityGML elements in the Standard. An example of rule is the relation between a main building and its building parts. This paper uses a similar approach, but it is more general, because it provides a set of constraint templates which can be instan-

tiated for obtaining a richer set of constraints depending on domain-specific information which exceeds the default specification of CityGML. Moreover, the author proposes an ad hoc implementation of the provided constraints through a Java Tool, where the constraints instantiated from the provided templates are automatically translated into SQL spatial queries.

In [17] the author provides a set of domain-specific constraints for a Climate City Campus Database described using CityGML. Examples of constraints are the distance between buildings and trees, or between aquatic plant and water. Such constraints are provided in OCL and translated into spatial queries for Oracle.

The use of OCL [11] for the specification of spatial constraints has been investigated also in [4] where the authors try to integrate the 9 Intersection Model into OCL. The obtained model is called OCL_{9IM} and provides an expressive language adapted to precisely model alphanumeric and topological constraints. They also investigate the possibility to translate OCL_{9IM} into SQL by providing an extension of the the tool named OCL2SQL [3].

The problem of consistency validation between semantic and geometry in CityGML is highlighted also in [9] where the author provides an overview of CityGML, its concepts, applications, impacts, and future developments.

3 Geometric Model

In order to define spatial integrity constraints on data models specified using UML, the reference geometric model used here is the ISO 19107 standard (Spatial Schema) which has also been integrated in the ISO 19136 (GML). This section briefly introduces the spatial data types of GML that are considered in this paper and the set of topological relations existing between them defined in terms of the well-known 9-intersection model [6]; this formal definition of the relations is necessary for guiding the implementation of constraints in the next section.

3.1 Spatial Data Types

The geometric model of GML consists of primitives, which may be combined to form *complexes*, *composite* or *aggregate* geometries. For each dimension, there is a geometric primitive: a zero-dimensional object is a *Point*, a one-dimensional object is a *Curve*, a two-dimensional object is a *Surface*, and a three-dimensional object is a *Solid*.

This paper concentrates on a subset of 3D datatypes of GML which are the most relevant for 3D city modeling, also with respect to CityGML and

Inspire. In particular, it considers polyhedral surfaces, multi-surfaces and solid objects.

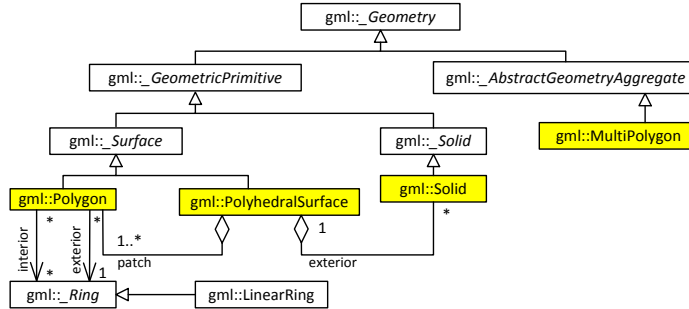


Figure 4: Reference spatial data types hierarchy.

Definition 3.1 (Abstract Surface). A surface is a 2-dimensional object. A simple surface may consist of a single patch that is associated with one exterior boundary and zero or more interior boundaries. The boundary of a simple surface is the set of closed curves corresponding to its exterior and interior boundaries [10].

In the Standard the class *Surface* is abstract, the only instantiable subclasses considered in this paper are the ones defined by OGC [12]: *Polygon* and *PolyhedralSurface*.

Definition 3.2 (Polygon). A Polygon is a planar surface defined by one exterior boundary and zero or more interior boundaries. Each interior boundary defines a hole in the polygon. The interior and exterior boundaries are defined by LinearRing such that the interior boundaries have an opposite orientation w.r.t. the exterior one.

Notice that a *Polygon* is valid if it is topologically closed, no two rings in the boundary cross and the rings in the boundary may intersect at a point but only as a tangent, its interior is a connected set of points, and the exterior with one or more holes is not connected [12].

Definition 3.3 (PolyhedralSurface). A PolyhedralSurface is a contiguous collection of Polygons (called patches), which share common boundary segments. For each pair of polygons that touch, the common boundary shall be expressible as a finite collection of LineStrings. Each such LineString shall be part of the boundary of at most 2 polygon patches.

Without loss of generality, the following simplification is introduced w.r.t. the standard: no holes are admitted in a patch and adjacent coplanar patches are not admitted, since they can be replaced by the patch obtained by merging them.

Definition 3.4 (Abstract Solid). A solid is a 3-dimensional object whose extent is defined by the boundary surfaces. In particular, these surfaces shall be organized into one set of surfaces for each boundary component of the solid, and each of these shells shall be a cycle [10].

In this paper a *Solid* is defined by a closed *PolyhedralSurface*. Notice that the *Solid* objects considered in this paper have no holes (like in CityGML), this is justified by the fact that only the external shell of a building is considered.

Definition 3.5 (MultiPolygon). A MultiPolygon is an unstructured set of polygons. No further constraints are defined for a MultiPolygon element.

MultiPolygon can be considered a generalization of *PolyhedralSurface*, since no particular constraints are required.

3.2 Topological Relations

The 9-intersection model [6] is the most common model for binary topological relations. It defines the topological relation R existing between two objects A and B considering the intersection between its interior (A°), boundary (∂A) and exterior (A^-).

$$R(A, B) = \begin{pmatrix} A^\circ \cap B^\circ & A^\circ \cap \partial B & A^\circ \cap B^- \\ \partial A \cap B^\circ & \partial A \cap \partial B & \partial A \cap B^- \\ A^- \cap B^\circ & A^- \cap \partial B & A^- \cap B^- \end{pmatrix}$$

In [5] the authors generalize the case to a 3D space. In order to do so, a major assumption is made which is valid in the context of this paper: the interior, boundary and exterior of the objects are simple connected such that the volume's boundary separate the interior from the exterior. Since the volume's boundary must be simply connected, the volume cannot have any holes in its interior. The authors further exclude holes of the type of a doughnut. Therefore, a 3D volume exposes the same properties as a 2D region.

As stated in the previous section, the paper concentrates only on 3D data types which are more useful in the description of city models: solid (denoted as V) and surfaces (denoted as S). Table 1 reports the formal

Table 1: Definition of the possible 3D topological relations between two solids (V/V), a solid and a surface (V/S), a surface and a solid (S/V) or two surfaces (S/S). Topological relations are denoted as: disjoint (DJ), touch (TC), in (IN), contains (CN), equal (EQ), overlap (OV). Only for the relation IN also the case curve/surface is considered (C/S).

Rel.	Definition	Geom.	Matrix Pattern
DJ	$A \cap B = \emptyset$	V/V, S/S V/S, S/V	$FFT - FFT - TTT$
TC	$(A^\circ \cap B^\circ = \emptyset) \wedge$ $(A \cap B \neq \emptyset)$	V/V	$FFT - FTT - TTT$
		V/S	$FFT - T** - **T$ $FFT - FTT - T*T$
		S/V	$FT* - F** - T*T$ $FFT - FT* - TTT$
		S/S	$FTT - *** - T*T$ $FFT - T** - T*T$ $FFT - FT* - T*T$
IN	$(A \cap B = A) \wedge$ $(A^\circ \cap B^\circ \neq \emptyset)$	V/V	$TFE - T*F - TTT$
		S/S	$TFE - **F - TTT$
		S/V, C/S	$T*F - **F - TTT$
CN	$(A \cap B = B) \wedge$ $(A^\circ \cap B^\circ \neq \emptyset)$	V/V, S/S	$TTT - F*T - FFT$
		V/S	$T*T - **T - FFT$
EQ	$A = B$	V/V, S/S	$TFE - FTF - FFT$
OV	$(A^\circ \cap B^\circ \neq \emptyset) \wedge$ $(A \cap B \neq A) \wedge$ $(A \cap B \neq B)$	V/V	$TTT - TTT - TTT$
		S/S	$T*T - *** - T*T$
		V/S	$T*T - T*T - T*T$
		S/V	$TTT - *** - TTT$

definition of each topological relation, together with the specification of the pair of geometric types to which it applies and the corresponding template of the 9-intersection matrix. The symbol T means that the intersection is not empty, F that the intersection is empty and $*$ that the intersection can be indifferently empty or not empty.

4 Spatial Constraint Templates

This section applies the approach proposed in [14] to city data models by defining OCL templates also for UML classes with spatial attributes having 3D geometric types. The OCL templates have a fixed logical structure, but have parameters that allow one to change the involved (constrained and constraining) classes, the spatial attributes, the required topological relations, the optional functions to be applied on geometries, the optional selection on constrained or constraining class.

The fundamental difference between the template approach proposed in this paper and the generic spatial OCL approach reported in Section 2 is the following: instead of augmenting OCL with new spatial operators and implement a generic translator of OCL expressions into a chosen target technology (i.e., SQL), a set of spatially oriented OCL templates with parameters are defined and an optimized implementation of them is specified for a set of target technologies.

Spatial constraint templates are defined by combining a logical structure with a topological relation. Two types of logical structure are considered: the existential (the most used) and the universal structure. An existential spatial constraint requires that, given an object x belonging to the constrained class X , there exists an object y of the constraining class Y such that the given topological relation (or disjunction of topological relations) is satisfied.

Definition 4.1 (SC_{\exists}). Let X be a constrained class with a spatial attribute g , Y be a constraining class with a spatial attribute f and rel_1, \dots, rel_n a disjunction of relation. An **existential spatial constraint** (SC_{\exists}) requires that for each object x of X there exists an object y of Y such that one of the relation rel_1, \dots, rel_n is satisfied between $x.g$ and $y.f$. It allows also to express selection conditions for X or Y or both. The selection σ_2 regarding Y can contain attributes of Y and also attributes of X , while the selection σ_1 on X can contain only attributes of X . When selection conditions are specified, the constraint is applied only to those objects of X that satisfies the selection σ_1 . Moreover, only those objects of Y that satisfies the selection σ_2 can be considered for testing the required topological relations. Finally some spatial functions $s_1()$ and $s_2()$ can be applied to the attributes f and g . They are optional, thus they can be left **null**.

$SC_{\exists}(X, \sigma_1(x), g, s_1(), \{rel_1 \dots rel_n\}, Y, \sigma_2(x, y), f, s_2())$
context X
inv: $\sigma_1(\mathbf{self}) \text{ implies } (Y.\mathbf{allinstances} \rightarrow$ $\quad \mathbf{exists}(a : Y \mid \sigma_2(\mathbf{self}, a) \wedge$ $\quad \quad (\mathbf{check}(\mathbf{self}.g.s_1(), \{rel_1, \dots, rel_k\}, a.f.s_2()))))$

where the expressions $\sigma_1(\mathbf{self})$ and $\sigma_2(\mathbf{self}, a)$ indicate the expressions that are obtained by replacing x with \mathbf{self} and y with a in $\sigma_1(x)$ and $\sigma_2(x, y)$, respectively; moreover, they can be **true** if no sections are required. Finally, if $s_1()$ (or $s_2()$) is null the function is not inserted in the expression, i.e. $g.s_1() \rightarrow g$ (or $f.s_2() \rightarrow f$).

The predicate **check** verifies that one of the topological relation $\{rel_1, \dots, rel_k\}$ is satisfied between the two given geometries ($a.rel_i(b)$ semantics is defined in Tab. 1):

$$\mathbf{check}(a, \{rel_1, \dots, rel_k\}, b) \equiv_{def} a.rel_1(b) \text{ or } \dots \text{ or } a.rel_k(b).$$

Example 4.1. Referring to the examples in Sec. 1.1 and the CityGML model, a basic existential topological constraint can be defined between all instances of the class **BuildingPart** and an instance of the class **Building**. More specifically, given an instance x of **BuildingPart** there exists an instance a of **Building** such that x touches a :

$$SC_{\exists}(\mathbf{BuildingPart}, \mathbf{true}, lod3Solid, \mathbf{null}, \{\mathbf{TC}\}, \\ \mathbf{Building}, \mathbf{true}, lod3Solid, \mathbf{null})$$

Accordingly with CityGML, this example assumes that both classes have a geometric property called *lod3Solid* which represents its extent. Fig. 5 shows a graphical representation of such constraint.

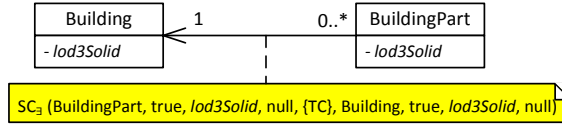


Figure 5: Example of existential spatial constraint between a main building and its parts.

The example below includes also some selection conditions.

Example 4.2. With reference to CityGML, an **Opening** shall be connected to a **BoundarySurface**. However, not all kinds of boundary surface can contain a window or a door; in particular, it is reasonable to assume that only an instance of **RoofSurface** or of **WallSurface** can contain a window or a door. This constraint can be represented using a selection on the surface type as follows:

$$SC_{\exists}(\mathbf{Opening}, \mathbf{true}, lod3MultiSurface, \mathbf{null}, \{\mathbf{IN}\}, \\ _BoundarySurface, \\ y.IsTypeOf(\mathbf{RoofSurface}, \mathbf{WallSurface}), \\ lod3MultiSurface, \mathbf{null})$$

Finally, the following example includes a spatial function.

Example 4.3. *The existential spatial constraint on boundary is particularly useful for expressing the constraint existing between an opening (e.g., window) and the solid representing a building. In particular, in the simple model discussed in Sec. 1.1 where windows are represented as surfaces that lies on a building solid, it is necessary to ensure that such surface is contained in the solid boundary.*

$$SC_{\exists}(\textit{Window}, \textit{true}, \textit{surface}, \textit{null}, \{\textit{IN}\}, \\ \textit{Building}, \textit{true}, \textit{solid}, \textit{boundary}()).$$

The universal logical structure replaces the existential quantification with a universal one [1][2].

Definition 4.2 (SC_{\forall}). Let X be a constrained class with a geometric attribute g , Y be a constraining class with a geometric attribute f and rel_1, \dots, rel_n a disjunction of relations. A **universal spatial constraint** requires that one of the topological relations rel_1, \dots, rel_n exists between the geometry of the constrained object $x.g$ and the geometry $y.f$ of all the objects y of the constraining class Y . Selection conditions and spatial functions can be applied also for this constraint, with the same meaning illustrated for SC_{\exists} .

The SC_{\forall} constraint is meaningful only for some kinds of topological relations, for instance *disjoint* or *touch*.

$SC_{\forall}(X, \sigma_1(x), g, s_1(), \{rel_1 \dots rel_n\}, Y, \sigma_2(x, y), f, s_2())$
context X
inv: $\sigma_1(\textit{self}) \textit{implies}(Y.\textit{allinstances} \rightarrow$ $\textit{forall}(a : Y \textit{self} = a \vee$ $(\textit{check}(\textit{self}.g.s_1(), \{rel_1, \dots, rel_k\}, a.f.s_2()))$

Example 4.4. *A typical usage for the universal spatial constraint is for the establishing that all buildings has to be disjoint or in touch:*

$$SC_{\forall}(\textit{Building}, \textit{true}, \textit{lod3Solid}, \textit{null}, \{\textit{TC} | \textit{DJ}\}, \\ \textit{Building}, \textit{true}, \textit{lod3Solid}, \textit{null})$$

In some cases it is necessary to specify a spatial constraint based on an association that links the constrained class and the constraining one. The following template deals with these cases.

Definition 4.3 (SC_{\forall}^r). Let X be a constrained class with a geometric attribute g , Y be a constraining class with a geometric attribute f , rel_1, \dots, rel_n be a disjunction of relation, and r_1, \dots, r_n be a chain of association roles that links the class X to the class Y . An **universal spatial constraint with a binding to a chain of roles** considers as available objects of the constraining class, only the objects that can be reached by the constrained object through the chain of roles $r_1 \dots r_n$.

The extension of the universal constraint to include a chain of association roles can be trivially deduced from this case.

$SC_{\forall}^r(X, \sigma_1(x), (r_1, \dots, r_n), g, s_1(), \{rel_1 \dots rel_n\},$ $Y, \sigma_2(x, y), f, s_2())$
context X inv: $\sigma_1(\text{self}) \text{ implies}(\text{self}.r_1 \rightarrow$ $\text{forall}(b_2 b_2.r_2 \rightarrow \dots \rightarrow \text{forall}(b_n b_n.r_n \rightarrow$ $\text{forall}(a : Y \sigma_2(\text{self}, a) \text{ implies}(\text{check}(\text{self}.g.s_1(), \{rel_1, \dots, rel_k\}, a.f.s_2())))) \dots)$

Example 4.5. *The existential spatial constraint with a binding to a chain of association roles can be used to model the conditions characterizing the openings in CityGML. In particular, let us consider the model in which each window (or door) of a building has to be embraced inside the outer boundary of the building itself, see Fig. 3. This condition can be represented stating this pair of constraints:*

$$\begin{aligned}
& SC_{\forall}^r(\text{Building}, \text{true}, (\text{boundedBy}, \text{opening}), \\
& \quad \text{lod3Solid}, \text{boundary}(), \{\text{CN}\}, \\
& \quad \text{_Opening}, \text{true}, \text{lod3MultiSurface}, \text{boundary}()) \\
& SC_{\forall}^r(\text{Building}, \text{true}, (\text{boundedBy}, \text{opening}), \\
& \quad \text{lod3Solid}, \text{boundary}(), \{\text{TC}\}, \\
& \quad \text{_Opening}, \text{true}, \text{lod3MultiSurface}, \text{null})
\end{aligned}$$

5 Template Implementation

This section presents the implementation of the constraint templates with reference to a vector model, which represents an abstract description of a current spatial DBMS able to deal with 3D data at some extent. The implementation on PostGIS of one template is also shown as real case example.

5.1 Vector Model

This paper aims to test the feasibility of representing and validating 3D spatial data in current GIS systems (the experiments adopt PostGIS 2.0 on PostgreSQL 9.4). To this purpose, this section presents a framework characterized by a discrete representation of solids, based on polyhedral surfaces of the geometric model of the Simple Feature Access for SQL (SFA) (OGC standard implemented in PostGIS), and a set of basic operations, that are necessary for the evaluation of topological relations between solids and between solids and surfaces. These operations have a corresponding implementation in current GIS systems or can be implemented using procedural language provided by them (in the experiments we implemented some basic operations using pl/pgsql of PostgreSQL 9.4). In order to simplify the presentation and focus on the validation approach, we consider only surfaces implemented as: polygons with no holes, multipolygons as set of polygons with no holes, and polyhedral surfaces as set of polygons with no holes with the known constraints presented in Def. 3.3. The extension to polygons with holes is possible, but is not presented in this paper. The following definitions formalize the framework.

Definition 5.1 (Basic vector types). In Table 2 some vector types are defined. They are a basis for defining the representation of spatial types.

Table 2: Basic vector types.

Vector type	Description
<i>vertex</i> v	It is a tuple of finite numbers representing a 3D coordinate: $v = (x, y, z)$.
<i>segment</i> $s (v_1, v_2)$	It is a pair of vertices and represents the segment obtained by considering the linear interpolation between them.
<i>ring</i> $r (v_1, \dots, v_n)$	It is a list of vertices, its linear interpolation represents a ring ($v_1 = v_n$).
<i>patch</i> $p (v_1, \dots, v_n)$	It is a finite part of a plane whose boundary is defined by a ring.

Definition 5.2 (Vector representation in 3D). Given a geometry g , its vector representation $VR(g)$ is defined as follows (v denotes a generic vertex, s a generic segment, p a generic patch and r a generic ring):

- If $g \in Polygon$ then $VR(g) = (v_1, \dots, v_n)$ with $n > 1$.
The polygon is planar, thus the vertices of $VR(g)$ are coplanar. The

linear interpolation between any two consecutive vertices v_i, v_{i+1} is a segment s_i and the sequence of segments (s_1, \dots, s_{n-1}) is a ring. Therefore alternative discrete representations can be a list of segments: $VR'(g) = (s_1, \dots, s_m)$ with $m = n - 1$ and $s_i = (v_i, v_{i+1})$ or a ring $VR''(g) = r$.

- If $g \in PolyhedralSurface$ then $VR(g) = ((v_{1,1}, \dots, v_{1,n_1}), \dots, (v_{k,1}, \dots, v_{k,n_k}))$ with $n_i > 2$ and $k > 0$, where each list of vertices is a ring representing a polygon without holes. Using the patch definition, the discrete representation of a *PolyhedralSurface* can also be described as list of patches: $VR'(g) = (p_1, \dots, p_k)$ with $k > 0$ and where each p_i is a planar patch defined by the ring $r_i = (v_{i,1}, \dots, v_{i,n_i})$.
- If $g \in MultiPolygon$ then $VR(g) = ((v_{1,1}, \dots, v_{1,n_1}), \dots, (v_{k,1}, \dots, v_{k,n_k}))$ with $n_i > 2$ and $k > 0$, where each list of vertices is a ring representing a polygon without holes. Similarly to what have been proposed for *PolyhedralSurface*, a *MultiPolygon* can also be described as list of patches: $VR'(g) = (p_1, \dots, p_k)$ with $k > 0$ where each p_i is a polygon.
- If $g \in Solid$ then $VR(g) = ((v_{1,1}, \dots, v_{1,n_1}), \dots, (v_{k,1}, \dots, v_{k,n_k}))$ with $n_i > 2$ and $k > 3$. As in previous cases a *Solid* can also be described as list of patches: $VR'(g) = (p_1, \dots, p_k)$ with $k > 0$ and where each p_i is a planar patch. The set of patches must represent a polyhedral surface which is *simple*, i.e. it has no self-intersections and is a *cycle*, i.e. it has empty boundary.

Definition 5.3 (Basic predicates and operations). For each operation it is specified: the domain (the set of objects where the operation applies), one or more domains for parameters and the target domain. The possible domains are: *vertex*, *segment*, *ring*, *patch* (referenced together as **primitive**), *Polygon*, *MultiPolygon*, *PolyhedralSurface* (referenced together as **surface**) and **solid**; the domain **geometry** is the union of all previous domains.

Operations

- $vert : geometry \rightarrow \mathcal{P}(vertex)^1$, $g.vert()$ returns the set of vertices defining the geometry g , for *surface* and *solid* it can be derived from $VR(g)$.
- $seg : geometry \rightarrow \mathcal{P}(segment)$, $g.seg()$ returns the set of segments defining the geometry g , for *surface* and *solid* it can be derived from $VR'(g)$.

¹The symbol $\mathcal{P}(S)$ denotes the power set of the set S .

- $bnd : segment \rightarrow \mathcal{P}(vertex)$, $s.bnd()$ returns the set containing the start and end point of the segment s .
- $bnd : patch \cup surface \rightarrow \mathcal{P}(segment)$, $g.bnd()$ returns the set of segments defining the boundary of the patch/surface g .
- $bnd : solid \rightarrow polyhedral\ surface$, $sd.bnd()$ returns the polyhedral surface defining the boundary of the solid sd .
- $pat : surface \cup solid \rightarrow \mathcal{P}(patch)$, $sf.pat()$ returns the set of patches defining the polyhedral surface sf or the solid, i.e. $sf.pat() = VR'(sf)$.
- $intSeg : surface \rightarrow \mathcal{P}(segment)$, $sf.intSeg()$ returns the set of segments defining the boundaries of the patches $sf.pat()$ but that do not belong to $sf.bnd()$.
- $intVert : surface \rightarrow \mathcal{P}(segment)$, $sf.intVert()$ returns the set of vertices belonging to the rings of the patches $sf.pat()$ but that do not belong to segments of $sf.bnd()$.
- $ray_3 : vertex \cup segment \times solid \rightarrow integer$, $g.ray_3(sd)$ returns the number of patches of $sd.pat()$ that are intersected by the semi-straight line starting from $g.start()$ and passing through $g.end()$ (excluding the possible intersection at $g.start()$); when g is a vertex v the semi-straight line starting in $v = (x_v, y_v, z_v)$ with equation $y = y_v$, $z = z_v$ and $x > x_v$ is considered.

Predicates

- $eq_3 : geometry \times geometry \rightarrow boolean$, $g.eq_3(g_0)$ tests the equality between two geometries; two geometries are equal only if they have identical vector representation, i.e. $g.eq_3(g_0) \equiv VR(g) = VR(g_0)$.
- $cnt_3 : segment \times vertex \rightarrow boolean$, $s.cnt_3(v)$ tests the containment between a vertex v and the interior of a segment s : $v \in I(s)$.
- $int_3 : patch \times segment \rightarrow boolean$, $p.int_3(s)$ tests the intersection between the interior of a patch and the interior of a segment: $dim(I(p) \cap I(s)) = 0$. If the intersection has dimension 1, it returns false.
- $cnt_3 : patch \times vertex \rightarrow boolean$, $p.cnt_3(v)$ tests the inclusion between a vertex v and the interior of a patch p : $v \in I(p)$.

- $int_3 : patch \times patch \rightarrow boolean$, $p_1.int_3(p_2)$ tests the intersection between the interior of two patches: if $dim(I(p_1) \cap I(p_2)) = 1$ it returns true, otherwise it returns false.
- $cop : patch \cup segment \times segment \rightarrow boolean$, $p.cop(s)$ ($s_0.cop(s)$) tests the coplanarity between the patch p (or the segment s_0) and the segment s .
- $cop : patch \times patch \rightarrow boolean$, $p.cop(p_0)$ tests the coplanarity between the patches p and p_0 .
- $\langle rel \rangle_2 : geometry \times geometry \rightarrow boolean$, $g.\langle rel \rangle_2(g_0)$ tests the topological relation $\langle rel \rangle \in \{dj, tc, in, cn, ov\}$ between the projection of the geometries g and g_0 on the plane where both geometries lie.

We show hereby how the set of topological relations presented in Section 3 can be implemented using the operations and predicates introduced above and the vector representation of the geometries presented in Def. 5.2. This allows then to focus only on the implementation of the basic vector operations and predicates. The implementation of functions for topological relation tests changes with respect to the geometric type of objects and many different cases have to be considered; in this paper we show only some of them, in particular, those ones that involve solid geometries.

Proposition 5.1 (Tests implementation). *Given the basic operations and predicates presented in Def. 5.3 the implementation of the following topological relation tests can be formally defined as shown in Tab. 3. Due to space limitations, the analysis is limited to the tests that are necessary for the spatial integrity constraint examples shown in the previous sections.*

- *curve in surface: considering the matrix for in, $[T * F - ** F - TTT]$, when applied to rings and surfaces in the 3D space, only the interior-interior intersection, $INT^{\circ\circ}(cv, sf)$, and the interior-exterior intersection, $INT^{\circ-}(cv, sf)$, need to be tested (i.e. the matrix can be reduced to $[T * F - *** - ***]$).*
- *surface disjoint surface: the matrix in 3D is simplified and the necessary tests are reduced to $[FF * - FF * - ***]$.*
- *surface in surface: the matrix in 3D is simplified and the necessary test is reduced to $[* * F - T * * - ***]$. The “T” in the second row is necessary to exclude equals.*

- *surface touches surface: the matrix in 3D is simplified and the necessary test is reduced to $[F * T - * * * - * * *] \vee [F * * - T * * - * * *] \vee [F * * - * T * - * * *]$.*
- *solid disjoint solid: considering the matrix for disjoint ($[FFT - FFT - TTT]$) when applied to solids in the 3D space, only the interior-interior intersection ($int^{\circ\circ}(sd_1, sd_2)$) and the boundary-boundary intersection ($int^{\partial\partial}(sd_1, sd_2)$) need to be tested (i.e., the matrix is reduced to $[F * * - * F * - * * *]$).*
- *solid touches solid: the matrix in 3D is simplified and the necessary test is reduced to $[F * * - * T * - * * *]$.*

Proof. In order to prove that the tests for a relation r on type t_1/t_2 are implemented correctly, we show that the implementation proposed in Tab. 3 detects all the possible scenarios that correspond to the existence of the relation r in 3D space considering objects of types t_1/t_2 . The possible scenarios are presented in Tables 4, 5 , 6 and 7.

- *cv in sf:* this relation occurs when $INT^{\circ\circ}(cv, sf) \wedge \neg INT^{\circ-}(cv, sf)$. $INT^{\circ\circ}(cv, sf)$ is true if at least one segment of cv intersects the interior of sf ; sufficient conditions for obtaining this result are produced in the following possible scenarios of Tab. 4: cells (1, 1), (3, 2), (5, 2), (6, 2), (6, 4), (8, 1) ,(8, 2), (8, 3), (10, 2) and (10, 3). In the proposed test (first row of Tab. 3) scenarios (10, 2) and (10, 3) are covered by formula at line 1., (8, 1) ,(8, 2) and (8, 3) by formula at line 2., (6, 2), (6, 4) by formula at line 3., (5, 2) by formula at line 4., (3, 2) and (1, 1) by formula at line 5.
 $\neg INT^{\circ-}(cv, sf)$ is true if all segments of cv are contained in (or a equal to) some primitives of sf ; sufficient conditions for obtaining this result are produced in the following possible scenarios of Tab. 4: cells (8, 1), (8, 2), (9, 1), (9, 2) and (10, 2). In the proposed test (first row of Tab. 3) scenarios (10, 2) is covered by formula at line 6. and (8, 1), (8, 2), (9, 1), (9, 2) are covered by formula at line 7.
- *sf₁ disjoint sf₂:* this relation requires to test: $INT^{\circ\circ}(sf_1, sf_2)$, $INT^{\partial\partial}(sf_1, sf_2)$ and $INT^{\circ\partial}(sf_1, sf_2)$. $INT^{\circ\circ}(sf_1, sf_2)$ is true if exists at least one primitive covering the interior of sf_1 intersects one primitive covering the interior of sf_2 ; sufficient conditions for obtaining this result are produced in the following possible scenarios of Tab. 5: cells (1, 1), (5, 2), (9, 2), (13, 1), (13, 2), (13, 3), (17, 2), (19, 2), (19, 3), (21, 1), (21, 2), (21, 3), (24, 2) and (24, 3). In the proposed test (second row of Tab. 3)

Table 3: Implementation of topological relation tests.

Relation	#L	Test
<i>in</i> <i>curve/surf</i> $IN(cv, sf)$		$IN(cv, sf) \equiv INT^{\circ\circ}(cv, sf) \wedge \neg INT^{\circ-}(cv, sf).$ $INT^{\circ\circ}(cv, sf) \equiv$ 1. $\exists s \in cv.seg() (\exists p \in sf.pat() (p.int_3(s) \vee (p.cop(s) \wedge \neg p.dj_2(s) \wedge \neg p.tc_2(s)))) \vee$ 2. $\exists s_0 \in sf.intSeg() (s.cop(s_0) \wedge \neg s.dj_2(s_0) \wedge \neg s.tc_2(s_0)) \vee$ 3. $\exists v_0 \in sf.intVert() (s.cnt_3(v_0) \vee v_0.eq_3(s.start()) \vee v_0.eq_3(s.end())) \vee$ 4. $\exists v \in cv.vert() (v \notin cv.bnd() \wedge (\exists p \in sf.pat() (p.cnt_3(v)) \vee$ 5. $\exists s_0 \in sf.intSeg() (s_0.cnt_3(v) \vee \exists v_0 \in sf.intVert() (v.eq_3(v_0))))$ $\neg INT^{\circ-}(cv, sf) \equiv$ 6. $\forall s \in cv.seg() (\exists p \in sf.pat() (p.cop(s) \wedge p.cn_2(s)) \vee$ 7. $\exists s_0 \in sf.intSeg() (\cup sf.bnd() (s.eq_3(s_0) \vee (s_0.cop(s) \wedge s_0.cn_2(s))))$
<i>disjoint</i> <i>surf/surf</i> $DJ(sf_1, sf_2)$		$\neg INT^{\circ\circ}(sf_1, sf_2) \wedge \neg INT^{\partial\partial}(sf_1, sf_2) \wedge$ $\neg INT^{\partial\circ}(s_1, s_2) \wedge \neg INT^{\circ\partial}(s_1, s_2)$ $INT^{\circ\circ}(sf_1, sf_2) \equiv$ 8. $\exists p_1 \in sf_1.pat() (\exists p_2 \in sf_2.pat() (p_1.int_3(p_2) \vee (p_1.cop(p_2) \wedge \neg p_1.dj_2(p_2) \wedge \neg p_1.tc_2(p_2)))) \vee$ 9. $\exists s_1 \in sf_1.intSeg() (\exists p_2 \in sf_2.pat() (p_2.cop(s_1) \wedge \neg p_2.dj_2(s_1) \wedge \neg p_2.tc_2(s_1))) \vee$ 10. $\exists s_2 \in sf_2.intSeg() (\exists p_1 \in sf_1.pat() (p_1.cop(s_2) \wedge \neg p_1.dj_2(s_2) \wedge \neg p_1.tc_2(s_2))) \vee$ 11. $\exists v_1 \in sf_1.intVert() (\exists p_2 \in sf_2.pat() (p_2.cnt_3(v_1))) \vee$ 12. $\exists v_2 \in sf_2.intVert() (\exists p_1 \in sf_1.pat() (p_1.cnt_3(v_2))) \vee$ 13. $\exists s_1 \in sf_1.intSeg() (\exists s_2 \in sf_2.intSeg() (s_1.eq_3(s_2) \vee$ 14. $s_1.int_3(s_2) \vee (s_1.cop(s_2) \wedge \neg s_1.dj_2(s_2) \wedge \neg s_1.tc_2(s_2)))) \vee$ 15. $\exists v_1 \in sf_1.intVert() (\exists s_2 \in sf_2.intSeg() (s_2.cnt_3(v_1) \vee \exists v_2 \in sf_2.intVert() (v_1.eq_3(v_2)))) \vee$ 16. $\exists v_2 \in sf_2.intVert() (\exists s_1 \in sf_1.intSeg() (s_1.cnt_3(v_2)))$ $INT^{\partial\partial}(sf_1, sf_2) \equiv$ 17. $\exists s_1 \in sf_1.bnd() (\exists s_2 \in sf_2.bnd() (s_1.eq_3(s_2) \vee (s_1.cop(s_2) \wedge \neg s_1.dj_2(s_2))))$ $INT^{\partial\circ}(sf_1, sf_2) \equiv$ 18. $\exists s_1 \in sf_1.bnd() (\exists p_2 \in sf_2.pat() (p_2.int_3(s_1) \vee (p_2.cop(s_1) \wedge \neg p_2.dj_2(s_1) \wedge \neg p_2.tc_2(s_1))) \vee$ 19. $(p_2.cnt_3(s_1.start()) \vee p_2.cnt_3(s_1.end())) \vee$ 20. $\exists s_2 \in sf_2.intSeg() (s_1.cop(s_2) \wedge \neg s_1.dj_2(s_2)))$ $INT^{\circ\partial}(sf_1, sf_2) \equiv INT^{\partial\circ}(sf_2, sf_1)$
<i>touches</i> <i>surf/surf</i> $TC(sf_1, sf_2)$		$\neg INT^{\circ\circ}(sf_1, sf_2) \wedge (INT^{\partial\partial}(sf_1, sf_2) \vee INT^{\partial\circ}(s_1, s_2) \vee INT^{\circ\partial}(s_1, s_2))$ for $INT^{\circ\circ}(sf_1, fs_2)$, $INT^{\partial\partial}(sf_1, sf_2)$, $INT^{\partial\circ}(sf_1, sf_2)$ and $INT^{\circ\partial}(sf_1, sf_2)$ see previous test.
<i>in</i> <i>surf/surf</i> $IN(sf_1, sf_2)$		$\neg INT^{\circ-}(sf_1, sf_2) \wedge INT^{\partial\circ}(sf_1, sf_2)$ $\neg INT^{\circ-}(sf_1, sf_2) \equiv$ 21. $\forall p_1 \in sf_1.pat() (\exists p_2 \in sf_2.pat() (p_1.eq_3(p_2) \vee (p_1.cop(p_2) \wedge p_2.cn_2(p_1)))$ for $INT^{\partial\circ}(sf_1, sf_2)$ see the disjoint test on surfaces.
<i>disjoint</i> <i>solid/solid</i> $DJ(sd_1, sd_2)$		$\neg INT^{\circ\circ}(sd_1, sd_2) \wedge \neg INT^{\partial\partial}(sd_1, sd_2)$ $INT^{\circ\circ}(sd_1, sd_2) \equiv$ 22. $\exists v_1 \in sd_1.vert() (mod_2(v_1.ray_3(sd_2)) = 1) \vee \exists v_2 \in sd_2.vert() (mod_2(v_2.ray_3(sd_1)) = 1) \vee$ 23. $\exists p_1 \in sd_1.pat() (\exists p_2 \in sd_2.pat() (p_1.int_3(p_2)))$ where $mod_2(x)$ returns the rest of the division by 2. $INT^{\partial\partial}(sd_1, sd_2) \equiv INT^{\circ\circ}(sd_1.bnd(), sd_2.bnd())$ for $INT^{\circ\circ}(sd_1.bnd(), sd_2.bnd())$ see the disjoint test on surfaces.
<i>touches</i> <i>solid/solid</i> $TC(sd_1, sd_2)$		$\neg INT^{\circ\circ}(sd_1, sd_2) \wedge INT^{\partial\partial}(sd_1, sd_2)$ for $INT^{\circ\circ}(sd_1, sd_2)$ and $INT^{\partial\partial}(sd_1, sd_2)$ see previous test.

scenarios (1, 1) and (5, 2) are covered by formula at line 15., (9, 2) by formula at line 16., (13, 1) by formula at line 13., (13, 2) and (13, 3) by formula at line 14., (17, 2) by formula at line 12., (19, 2) and (19, 3) by formula at line 10., (21, 1), (21, 2) and (21, 3) by formula at line 8., (22, 2) by formula at line 11., (24, 2) and (24, 3) by formula at line 9.. Similarly for $INT^{\partial\partial}(sf_1, sf_2)$ the possible scenarios of Tab. 5 and 6 to be

considered as sufficient conditions are: cells (4, 1), (8, 2), (8, 4), (12, 2), (12, 4), (16, 1), (16, 2), (16, 3) and (16, 4). In the proposed test (second row of Tab. 3) scenarios (16, 1), (16, 2), (16, 3) and (16, 4) are covered directly by formula at line 17., the other scenarios are covered indirectly by the same formula since they can be detected as an instance of the touches relation indeed two touching segments are always coplanar and therefore they touches also in 2D (on the plane they lie). Finally for $INT^{\circ\partial}(sf_1, sf_2)$ the possible scenarios of Tab. 5 and 6 to be considered as sufficient conditions are: cells (3, 1), (7, 2), (7, 4), (11, 2), (11, 4), (15, 1), (15, 2), (15, 3), (15, 4), (23, 2), (25, 2) and (25, 3). In the proposed test (second row of Tab. 3) scenarios (25, 2) and (25, 3) are covered by formula at line 18., (23, 2) by formula at line 19., (15, 1), (15, 2), (15, 3) and (15, 4) by formula at line 20., the other scenarios are covered indirectly by the same formula for the same reasons explained before.

- sf_1 in sf_2 : this relation requires to test: $INT^{\circ-}(sf_1, sf_2)$ and $INT^{\partial\circ}(sf_1, sf_2)$. Only $INT^{\circ-}(sf_1, sf_2)$ has not been considered yet. $\neg INT^{\circ-}(sd_1, sd_2)$ is true if for all patches of sf_1 there exists a patch of sf_2 that contains it; sufficient conditions for obtaining this result are produced in the following possible scenarios of Tab. 5: and 6 cells (21, 1) and (21, 2). In the proposed test (forth row of Tab. 3) scenarios (21, 1) and (21, 2) are covered by formula at line 21..
- sd_1 disjoint sd_2 : this relation requires to test: $INT^{\circ\circ}(sd_1, sd_2)$ and $INT^{\partial\partial}(sd_1, sd_2)$. The second test can be reduced to $INT^{\circ\circ}(sd_1.bnd(), sd_2.bnd())$. $INT^{\circ\circ}(sd_1, sd_2)$ is true if the interior of sd_1 intersects at least one primitive defining sd_2 or viceversa; sufficient conditions for obtaining this result are produced in all the possible scenarios shown in Tab. 7. In the proposed test (fifth row of Tab. 3) scenarios (1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (6, 2) are covered by formula at line 22., while (3, 3) and (4, 3) are covered by formula at line 23.; finally also scenarios (2, 3) and (4, 3) are indirectly covered by formula at line 23., since given two solids if a segment of the polyhedral surface, that defines one of them, intersects the interior of the other solid, then also the patch bounded by it will intersect the other solid and viceversa.

Table 4: All possible scenarios to be considered to evaluate the existence of a topological relation between rings and surfaces. In the table the following symbols are used: v vertex, vi (vb) internal (or boundary) vertex of surfaces, s segment, si (sb) internal (or boundary) segments of surfaces, p is a patch, r is a ring, sf is a surface, sd is a solid.

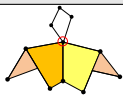
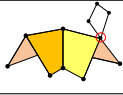
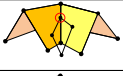


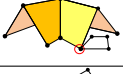
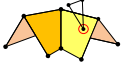
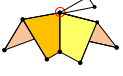
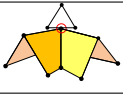
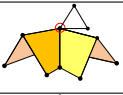
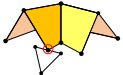
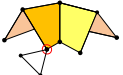
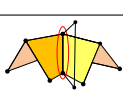
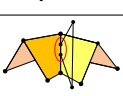
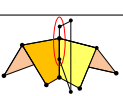
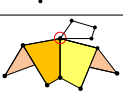
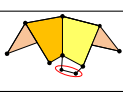
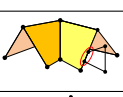
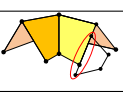
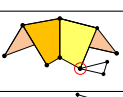
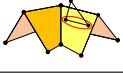
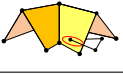
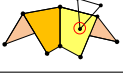
Primitives	1 EQ	2 IN/CN	3 OV	4 TC
1 $r.v - sf.vi$		—	—	—
2 $r.v - sf.vb$		—	—	—
3 $r.v - sf.si$	—		—	
4 $r.v - sf.sb$	—		—	
5 $r.v - sf.p$	—		—	
6 $r.s - sf.vi$	—		—	
7 $r.s - sf.vb$	—		—	
8 $r.s - sf.si$				
9 $r.s - sf.sb$				
10 $r.s - sf.p$	—			

Table 5: All possible scenarios to be considered to evaluate the existence of a topological relation between two surfaces (vertices and segments). For symbol interpretation see Tab. 4.


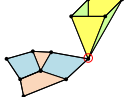
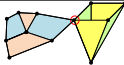




















Primitives		1 EQ	2 IN/CN	3 OV	4 TC
1	$sf_1.vi - sf_2.vi$		—	—	—
2	$sf_1.vi - sf_2.vb$		—	—	—
3	$sf_1.vb - sf_2.vi$	as 2	—	—	—
4	$sf_1.vb - sf_2.vb$		—	—	—
5	$sf_1.vi - sf_2.si$	—		—	
6	$sf_1.vi - sf_2.sb$	—		—	
7	$sf_1.vb - sf_2.si$	—		—	
8	$sf_1.vb - sf_2.sb$	—		—	
9	$sf_1.si - sf_2.vi$	—	as 5	—	as 5
10	$sf_1.si - sf_2.vb$	—	as 7	—	as 7
11	$sf_1.sb - sf_2.vi$	—	as 6	—	as 6
12	$sf_1.sb - sf_2.vb$	—	as 8	—	as 8
13	$sf_1.si - sf_2.si$				
14	$sf_1.si - sf_2.sb$				
15	$sf_1.sb - sf_2.si$	as 14	as 14	as 14	as 14
16	$sf_1.sb - sf_2.sb$				

Table 6: All possible scenarios to be considered to evaluate the existence of a topological relation between two surfaces (patches). For symbol interpretation see Tab. 4.

Primitives		1 EQ	2 IN/CN	3 OV	4 TC
17	$sf_1.p - sf_2.vi$	—		—	
18	$sf_1.p - sf_2.vb$	—		—	
19	$sf_1.p - sf_2.si$	—			
20	$sf_1.p - sf_2.sb$	—			
21	$sf_1.p - sf_2.p$				
22	$sf_1.vi - sf_2.p$	—	as 17	—	as 17
23	$sf_1.vb - sf_2.p$	—	as 18	—	as 18
24	$sf_1.si - sf_2.p$	—	as 19	as 19	as 19
25	$sf_1.sb - sf_2.p$	—	as 20	as 20	as 20

Table 7: Additional possible scenarios (w.r.t. the content of Tab. 5) to be considered to evaluate the existence of a topological relation between two solids. For symbol interpretation see Tab. 4, in addition vl is used to indicate the portion of 3D space occupied by the solid (i.e., the interior of the solid).

Primitives		1 EQ	2 IN/CN	3 OV	4 TC
1	$sd_1.vl - sd_2.v$	—		—	—
2	$sd_1.vl - sd_2.s$	—			—
3	$sd_1.vl - sd_2.p$	—			—
4	$sd_1.v - sd_2.vl$	—	as 1	—	—
5	$sd_1.s - sd_2.vl$	—	as 2	as 2	—
6	$sd_1.p - sd_2.vl$	—	as 3	as 3	—

5.2 SQL Implementation of Templates

In this subsection we show how each of the constraint template presented in Sec. 4 can be implemented in SQL. For each spatial constraint template SC_{∇}^r , we show the corresponding SQL template that is able to extract all the tuples representing objects of the constrained class that violate the constraint. The SQL template is instantiated and executed on a relational database implementing the UML classes in relational tables. The relational schema has been obtained by following some mapping rules that guarantee the ability of representing all possible states of the objects that describe an instance of the given UML classes. For example, each constrained class X (constraining class Y) is represented by one table T_X (T_Y) containing all its (non-multi) properties represented as columns with the same name of the properties.

In the following boxes for each constraint template the corresponding SQL template is shown. Parameters are written in italics, F_{s_1} and F_{s_2} are the corresponding functions in PostGIS of $s_1()$ and $s_2()$, respectively. The functions $F_{rel_i}(g, f)$ are the implementation in PL/pgSQL of the relation tests defined by Prop.5.1. They are based on the vector operations and predicates that have been defined in Def. 5.3. Many of them are already implemented in current GIS system (e.g., PostGIS), while some others require an ad hoc implementation; in particular, $sf.intSeg()$, $sf.intVert()$, $g.ray_3(P)$, $g.eq_3(g_0)$, $s.cnt_3(v)$, $p.int_3(s)$, $p.cnt_3(v)$, $p.cnt_3(v)$, $p_1.int_3(p_2)$, $p.cop(s)$ and $p.cop(p_0)$ have to be implemented. In the following subsection we present the implementation in PostGIS of each of them.

$SC_{\exists}(X, \sigma_1(x), g, s_1(), \{rel_1 ... rel_n\}, Y, \sigma_2(x, y), f, s_2())$
<p>SQL template</p> <pre> SELECT x.* FROM T_X as x WHERE $\sigma_1(x)$ AND NOT EXISTS (SELECT 1 FROM T_Y as y WHERE $\sigma_2(x, y)$ AND ($F_{rel_1}(F_{s_1}(x.g), F_{s_2}(y.f))$) OR ... OR $F_{rel_n}(F_{s_1}(x.g), F_{s_2}(y.f))$)) </pre>

$SC_{\forall}(X, \sigma_1(x), g, s_1(), \{rel_1 .. rel_n\}, Y, \sigma_2(x, y), f, s_2())$
SQL template SELECT x.* FROM T_X as x WHERE $\sigma_1(x)$ AND EXISTS (SELECT 1 FROM T_Y as y WHERE $\sigma_2(x, y)$ AND NOT($F_{rel_1}(F_{s_1}(x.g), F_{s_2}(y.f))$) OR ... OR $F_{rel_n}(F_{s_1}(x.g), F_{s_2}(y.f))$))

$SC_{\forall}^r(X, \sigma_1(x), (r_1, \dots, r_n), g, s_1(), \{rel_1 .. rel_n\}, Y, \sigma_2(x, y), f, s_2())$
SQL template SELECT x.* FROM T_X as x WHERE $\sigma_1(x)$ AND EXISTS (SELECT 1 FROM T_Y as y JOIN T_{n-1} as y_{n-1} ON $y_{n-1}.r_n = y.ID$... JOIN T_1 as y_1 ON $y_1.r_2 = y_2.ID$ WHERE $x.r_1 = y_1.ID$ AND $\sigma_2(x, y)$ AND NOT($F_{rel_1}(F_{s_1}(x.g), F_{s_2}(y.f))$) OR ...OR $F_{rel_n}(F_{s_1}(x.g), F_{s_2}(y.f))$))

Considering the Example 4.5 the query testing the first constraint of the example, obtained by apply the previous SQL template, has the following form:

SQL Query SELECT x.* FROM T_Building as x WHERE true AND EXISTS(SELECT 1 FROM V_Opening as y JOIN V_BoundarySurface as y_1 ON $y_1.opening=y.ID$ WHERE $x.boundedBy = y_1.ID$ AND true AND NOT($F_{CN}(ST_Boundary(x.lod3Solid),$ $ST_Boundary(y.lod3MultiSurface))$)
--

5.3 Implementation of basic operations and predicates in SQL

Each one of the following operations and predicates has to be implemented in PostGIS, since it is not currently present in the system: $sf.intSeg()$, $sf.intVert()$, $g.ray_3(sd)$, $g.eq_3(g_0)$, $s.cnt_3(v)$, $p.int_3(s)$, $p.cnt_3(v)$, $p_1.int_3(p_2)$, $g.cop(s)$ (where g can be a patch or a segment) and $p.cop(p_0)$.

However, some of them can be derived from the others, thus as first step we show the derived operations and predicates and then we show the implementation of the basic ones as SQL functions.

Derived operations

- $sf.intSeg()$ - The internal segments of a surface sf can be obtained by subtracting from the set of all segments of sf ($sf.seg()$) the set of segments representing its boundary ($sf.bnd()$):

$$sf.intSeg() \equiv sf.seg() \setminus sf.bnd()$$

- $sf.intVert()$ - The internal vertices of a surface sf can be obtained by subtracting from the set of all vertices of sf ($sf.vert()$) the set of vertices of the segments representing its boundary ($sf.bnd().vert()$):

$$sf.intVert() \equiv sf.vert() \setminus sf.bnd().vert()$$

- $p.cop(p_0)$ - The patch p_0 is contained in the patch p if at least two of its segments $s_1, s_2 \in p_0.seg()$ are coplanar with p .

$$p.cnt_3(v) \equiv \exists s_1 \in p_0.seg()(p.cop(s_1) \wedge \exists s_2 \in p_0.seg()(\neg s_1.eq_3(s_2) \wedge p.cop(s_2)))$$

Therefore, the basic operations and predicates are: $g.eq_3(g_0)$, $s.cnt_3(v)$, $p.int_3(s)$, $p.cnt_3(v)$, $p_1.int_3(p_2)$, $g.cop(s)$ and $g.ray_3(sd)$. In the following boxes we show for each of the basic operations and predicates the corresponding function implemented in PL/pgSQL.

g.eq₃(g₀)

SQL function

```
CREATE FUNCTION EQ_3 (geometry, geometry)
  RETURNS boolean AS $$
DECLARE ...
BEGIN
g1 := $1; g2 := $2;
IF NOT ST_EQUALS(g1,g2) THEN RETURN false; END IF;
i := 1;
FOR p IN SELECT geom FROM ST_DUMPPOINTS(g2) ORDER BY
  ST_X(geom),ST_Y(geom),ST_Z(geom) LOOP
  pt2[i] := p;
  i := i + 1;
END LOOP;
i := 1;
FOR pt1 IN SELECT geom FROM ST_DUMPPOINTS(g1) ORDER BY
  ST_X(geom),ST_Y(geom),ST_Z(geom) LOOP
  p := pt2[i];
  IF ST_X(pt1) <> ST_X(p) OR ST_Y(pt1) <> ST_Y(p)
    OR ST_Z(pt1) <> ST_Z(p) THEN
    RETURN false;
  ELSE
    i := i + 1;
  END IF;
END LOOP;
RETURN true;
END;
$$ LANGUAGE plpgsql;
```

s.cnt₃(v)

SQL function

```
CREATE FUNCTION S_CNT_V_3 (geometry, geometry)
  RETURNS boolean AS $$
DECLARE ...
BEGIN
s := $1; p := $2;
x0 := ST_X(p); y0 := ST_Y(p); z0 := ST_Z(p);
IF ST_X(p) < ST_XMIN(s) OR ST_X(p) > ST_XMAX(s) OR
  ST_Y(p) < ST_YMIN(s) OR ST_Y(p) > ST_YMAX(s) OR
  ST_Z(p) < ST_ZMIN(s) OR ST_Z(p) > ST_ZMAX(s) THEN
  RETURN false;
END IF;
p1 := ST_STARTPOINT(s); x1 := ST_X(p1); y1 := ST_Y(p1); z1 := ST_Z(p1);
p2 := ST_ENDPOINT(s); x2 := ST_X(p2); y2 := ST_Y(p2); z2 := ST_Z(p2);
IF (x2-x1) = 0 THEN deltax := 0; deltaz := 0;
ELSE deltax := (y2-y1)/(x2-x1); deltaz := (z2-z1)/(x2-x1);
END IF;
x3 := (x1+x2)/2 - 1; y3 := (y1+y2)/2 + deltax - 5;
z3 := (z1+z2)/2 + deltaz - 7;
a := (y2 - y1)*(z3 - z1) - (z2 - z1)*(y3 - y1);
b := -((x2 - x1)*(z3 - z1) - (z2 - z1)*(x3 - x1));
c := (x2 - x1)*(y3 - y1) - (y2 - y1)*(x3 - x1);
d := -(a*x1 + b*y1 + c*z1);
IF (abs(a*x0 + b*y0 + c*z0 + d) > 1012) THEN
  RETURN false;
END IF;
x3 := (x1+x2)/2 + 1; y3 := (y1+y2)/2 + deltax + 3;
z3 := (z1+z2)/2 + deltaz + 5;
a := (y2 - y1)*(z3 - z1) - (z2 - z1)*(y3 - y1);
b := -((x2 - x1)*(z3 - z1) - (z2 - z1)*(x3 - x1));
c := (x2 - x1)*(y3 - y1) - (y2 - y1)*(x3 - x1);
d := -(a*x1 + b*y1 + c*z1);
IF (abs(a*x0 + b*y0 + c*z0 + d) > 1012) THEN
  RETURN false;
END IF;
RETURN true;
END;
$$ LANGUAGE plpgsql;
```

SQL function - first part

```
CREATE FUNCTION P_INT_S_3 (geometry, geometry)
  RETURNS boolean AS $$
DECLARE ...
BEGIN
p := $1; s := $2;
p1 := ST_STARTPOINT(s);x1 := ST_X(p1);y1 := ST_Y(p1);z1 := ST_Z(p1);
p2 := ST_ENDPOINT(s);x2 := ST_X(p2);y2 := ST_Y(p2);z2 := ST_Z(p2);
IF (x2-x1) = 0 THEN deltay := 0; deltaz := 0;
ELSE deltay := (y2-y1)/(x2-x1); deltaz := (z2-z1)/(x2-x1);
END IF;
-- first plane
x3 := (x1+x2)/2 + 1;y3 := (y1+y2)/2 + deltay - 5;
z3 := (z1+z2)/2 + deltaz - 7;
a1 := (y2 - y1)*(z3 - z1) - (z2 - z1)*(y3 - y1);
b1 := -((x2 - x1)*(z3 - z1) - (z2 - z1)*(x3 - x1));
c1 := (x2 - x1)*(y3 - y1) - (y2 - y1)*(x3 - x1);
t1 := (a1*x1 + b1*y1 + c1*z1);
-- second plane
x3 := (x1+x2)/2 - 1;y3 := (y1+y2)/2 + deltay + 3;
z3 := (z1+z2)/2 + deltaz + 5;
a2 := (y2 - y1)*(z3 - z1) - (z2 - z1)*(y3 - y1);
b2 := -((x2 - x1)*(z3 - z1) - (z2 - z1)*(x3 - x1));
c2 := (x2 - x1)*(y3 - y1) - (y2 - y1)*(x3 - x1);
t2 := (a2*x1 + b2*y1 + c2*z1);
-- third plane
i := 1;
FOR pt IN SELECT geom FROM ST_DUMPPPOINTS(p) ORDER BY path[1] LOOP
  IF (i > 3) THEN EXIT; END IF;
  IF (i = 1) THEN x1 := ST_X(pt);y1 := ST_Y(pt);z1 := ST_Z(pt);END IF;
  IF (i = 2) THEN x2 := ST_X(pt);y2 := ST_Y(pt);z2 := ST_Z(pt);END IF;
  IF (i = 3) THEN x3 := ST_X(pt);y3 := ST_Y(pt);z3 := ST_Z(pt);END IF;
  i := i + 1;
END LOOP;
a3 := (y2 - y1)*(z3 - z1) - (z2 - z1)*(y3 - y1);
b3 := -((x2 - x1)*(z3 - z1) - (z2 - z1)*(x3 - x1));
c3 := (x2 - x1)*(y3 - y1) - (y2 - y1)*(x3 - x1);
t3 := (a3*x1 + b3*y1 + c3*z1);
det := (a1*b2*c3) + (b1*c2*a3) + (c1*a2*b3) -
        (c1*b2*a3) - (a2*b1*c3) - (a1*b3*c2);
IF (det = 0) THEN RETURN false; END IF;
```

SQL function - second part

```
xint := ((t1*b2*c3) + (b1*c2*t3) + (c1*t2*b3) -
         (c1*b2*t3) - (t2*b1*c3) - (t1*b3*c2))/det;
yint := ((a1*t2*c3) + (t1*c2*a3) + (c1*a2*t3) -
         (c1*t2*a3) - (a2*t1*c3) - (a1*t3*c2))/det;
zint := ((a1*b2*t3) + (b1*t2*a3) + (t1*a2*b3) -
         (t1*b2*a3) - (a2*b1*t3) - (a1*b3*t2))/det;
IF xint < ST_XMIN(s) OR xint > ST_XMAX(s) OR
   yint < ST_YMIN(s) OR yint > ST_YMAX(s) OR
   zint < ST_ZMIN(s) OR zint > ST_ZMAX(s) THEN
  RETURN false;
END IF;
pxy := ST_FORCE_2D(p);
gt1 := 'SRID='||ST_SRID(p)||';POLYGON(('; gt2 = gt1;
FOR pt IN SELECT geom FROM ST_DUMPPPOINTS(p) ORDER BY path[1] LOOP
  gt1 := gt1||ST_X(pt)||' '|| ST_Z(pt)||',';
  gt2 := gt2||ST_Y(pt)||' '|| ST_Z(pt)||',';
END LOOP;
gt1 := substring(gt1 from 1 for length(gt1)-1);
gt2 := substring(gt2 from 1 for length(gt2)-1);
gt1 := gt1 || '))'; gt2 := gt2 || '))';
pxz := ST_GEOMFROMEWKT(gt1); pyz := ST_GEOMFROMEWKT(gt2);
areaxy := 0; IF ST_ISVALID(pxy) THEN areaxy := ST_AREA(pxy); END IF;
areaxz := 0; IF ST_ISVALID(pxz) THEN areaxz := ST_AREA(pxz); END IF;
areayz := 0; IF ST_ISVALID(pyz) THEN areayz := ST_AREA(pyz); END IF;
IF areaxy > areaxz THEN
  IF areaxy > areayz THEN xc := xint; yc := yint; pch := pxy;
  ELSE xc := yint; yc := zint; pch := pyz;
  END IF;
ELSE
  IF areaxz > areayz THEN xc := xint; yc := zint; pch := pxz;
  ELSE xc := yint; yc := zint; pch := pyz;
  END IF;
END IF;
pint := ST_GEOMFROMEWKT('SRID='||ST_SRID(p)||';POINT('||xc||' '||yc||')');
RETURN ST_WITHIN(pint, pch);
END; $$ LANGUAGE plpgsql;
```

SQL function - first part

```
CREATE FUNCTION P_INT_S_3 (geometry, geometry)
  RETURNS boolean AS $$
  DECLARE ...
  BEGIN
    p := $1; v := $2;
    x0 := ST_X(v); y0 := ST_Y(v); z0 := ST_Z(v);
    i := 1;
    FOR pt IN SELECT geom FROM ST_DUMPPPOINTS(p) ORDER BY path[1] LOOP
      IF (i > 3) THEN EXIT; END IF;
      IF (i = 1) THEN x1 := ST_X(pt); y1 := ST_Y(pt); z1 := ST_Z(pt); END IF;
      IF (i = 2) THEN x2 := ST_X(pt); y2 := ST_Y(pt); z2 := ST_Z(pt); END IF;
      IF (i = 3) THEN x3 := ST_X(pt); y3 := ST_Y(pt); z3 := ST_Z(pt); END IF;
      i := i + 1;
    END LOOP;
    a1 := (y2 - y1)*(z3 - z1) - (z2 - z1)*(y3 - y1);
    b1 := -((x2 - x1)*(z3 - z1) - (z2 - z1)*(x3 - x1));
    c1 := (x2 - x1)*(y3 - y1) - (y2 - y1)*(x3 - x1);
    d1 := - (a1*x1 + b1*y1 + c1*z1);
    IF (abs(a1*x0 + b1*y0 + c1*z0 + d1) > 1012)) THEN
      RETURN false;
    END IF;
    pxy := ST_FORCE_2D(p);
    gt1 := 'SRID=' || ST_SRID(p)::text || ';POLYGON((';
    gt2 = gt1;
    FOR pt IN SELECT geom FROM ST_DUMPPPOINTS(p) ORDER BY path[1] LOOP
      gt1 = gt1||ST_X(pt)||' '||ST_Z(pt)||',';
      gt2 = gt2||ST_Y(pt)||' '||ST_Z(pt)||',';
    END LOOP;
    gt1 := substring(gt1 from 1 for length(gt1)-1);
    gt2 := substring(gt2 from 1 for length(gt2)-1);
    gt1 := gt1 || '))'; gt2 := gt2 || '))';
    pxz := ST_GEOMFROMEWKT(gt1); pyz := ST_GEOMFROMEWKT(gt2);
    areaxy := 0; IF ST_ISVALID(pxy) THEN areaxy := ST_AREA(pxy); END IF;
    areaxz := 0; IF ST_ISVALID(pxz) THEN areaxz := ST_AREA(pxz); END IF;
    areayz := 0; IF ST_ISVALID(pyz) THEN areayz := ST_AREA(pyz); END IF;
```


SQL function - second part

```
IF areaxy > areaxz THEN
  IF areaxy > areayz THEN xc := x0; yc := y0; pch := pxy;
  ELSE xc := y0; yc := z0; pch := pyz;
  END IF;
ELSE
  IF areaxz > areayz THEN xc := x0; yc := z0; pch := pxz;
  ELSE xc := y0; yc := z0; pch := pyz;
  END IF;
END IF;
vc := ST_GEOMFROMEWKT('SRID='||ST_SRID(p)||';POINT('||xc||' '||yc||')')
RETURN ST_WITHIN(vc, pch);
END; $$ LANGUAGE plpgsql;
```

SQL function - first part

```
CREATE FUNCTION P_INT_P_3 (geometry, geometry)
  RETURNS boolean AS $$
  DECLARE ...
  BEGIN
    p1 := $1; p2 := $2;
    i := 1;
    FOR pt IN SELECT geom FROM ST_DUMPPPOINTS(p1) ORDER BY path[1] LOOP
      IF (i > 3) THEN EXIT; END IF;
      IF (i = 1) THEN x1 := ST_X(pt); y1 := ST_Y(pt); z1 := ST_Z(pt); END IF;
      IF (i = 2) THEN x2 := ST_X(pt); y2 := ST_Y(pt); z2 := ST_Z(pt); END IF;
      IF (i = 3) THEN x3 := ST_X(pt); y3 := ST_Y(pt); z3 := ST_Z(pt); END IF;
      i := i + 1;
    END LOOP;
    a1 := (y2 - y1)*(z3 - z1) - (z2 - z1)*(y3 - y1);
    b1 := -((x2 - x1)*(z3 - z1) - (z2 - z1)*(x3 - x1));
    c1 := (x2 - x1)*(y3 - y1) - (y2 - y1)*(x3 - x1);
    d1 := - (a1*x1 + b1*y1 + c1*z1);
    i := 1;
    FOR pt IN SELECT geom FROM ST_DUMPPPOINTS(p2) ORDER BY path[1] LOOP
      IF (i > 3) THEN EXIT; END IF;
      IF (i = 1) THEN x1 := ST_X(pt); y1 := ST_Y(pt); z1 := ST_Z(pt); END IF;
      IF (i = 2) THEN x2 := ST_X(pt); y2 := ST_Y(pt); z2 := ST_Z(pt); END IF;
      IF (i = 3) THEN x3 := ST_X(pt); y3 := ST_Y(pt); z3 := ST_Z(pt); END IF;
      i := i + 1;
    END LOOP;
    a2 := (y2 - y1)*(z3 - z1) - (z2 - z1)*(y3 - y1);
    b2 := -((x2 - x1)*(z3 - z1) - (z2 - z1)*(x3 - x1));
    c2 := (x2 - x1)*(y3 - y1) - (y2 - y1)*(x3 - x1);
    d2 := - (a2*x1 + b2*y1 + c2*z1);
    IF (a1*b2)-(a2*b1) = 0 AND (a1*c2)-(a2*c1) = 0 AND
      (b1*c2)-(b2*c1) = 0 THEN
      RAISE NOTICE 'coplanar planes'; RETURN false;
    END IF;
```

SQL function - second part

```
-- intersecting planes: compute straight line of intersection
xs := ST_XMIN(p1);
IF (ST_XMIN(p2) < xs) THEN xs := ST_XMIN(p2); END IF;
xe := ST_XMAX(p1);
IF (ST_XMAX(p2) < xe) THEN xe := ST_XMAX(p2); END IF;
det := (b1*c2)-(b2*c1);
t1 := -d1 - a1*xs; t2 := -d2 - a2*xs;
ys := ((t1*c2)-(t2*c1))/det;
zs := ((b1*t2)-(b2*t1))/det;
t1 := -d1 - a1*xe; t2 := -d2 - a2*xe;
ye := ((t1*c2)-(t2*c1))/det;
ze := ((b1*t2)-(b2*t1))/det;
-- analyzing first patch
pxy := ST_FORCE_2D(p1);
gt1 := 'SRID=' || ST_SRID(p1)::text || ';POLYGON(('; gt2 = gt1;
FOR pt IN SELECT geom FROM ST_DUMPPPOINTS(p1) ORDER BY path[1] LOOP
  gt1 = gt1||ST_X(pt)||' '||ST_Z(pt)|| ',';
  gt2 = gt2||ST_Y(pt)||' '||ST_Z(pt)|| ',';
END LOOP;
gt1 := substring(gt1 from 1 for length(gt1)-1);
gt2 := substring(gt2 from 1 for length(gt2)-1);
gt1 := gt1 || '))';
gt2 := gt2 || '))';
pxz := ST_GEOMFROMEWKT(gt1);
pyz := ST_GEOMFROMEWKT(gt2);
areaxy := 0; IF ST_ISVALID(pxy) THEN areaxy := ST_AREA(pxy); END IF;
areaxz := 0; IF ST_ISVALID(pxz) THEN areaxz := ST_AREA(pxz); END IF;
areayz := 0; IF ST_ISVALID(pyz) THEN areayz := ST_AREA(pyz); END IF;
IF areaxy > areaxz THEN
  IF areaxy > areayz THEN xs0 := xs; ys0 := ys; xe0 := xe; ye0 := ye; pch := pxy;
  ELSE xs0 := ys; ys0 := zs; xe0 := ye; ye0 := ze; pch := pyz;
  END IF;
ELSE
  IF areaxz > areayz THEN xs0 := xs; ys0 := zs; xe0 := xe; ye0 := ze; pch := pxz;
  ELSE xs0 := ys; ys0 := zs; xe0 := ye; ye0 := ze; pch := pyz;
  END IF;
END IF;
ints := ST_GEOMFROMEWKT('SRID='||ST_SRID(p1)||
  ';LINSTRING('||xs0||' '||ys0||','||xe0||' '||ye0||')');
IF NOT (ST_CROSSES(pch,ints) OR ST_CONTAINS(pch,ints)) THEN RETURN false; END IF;
```

SQL function - third part

```
-- analyzing second patch
ints := ST_INTERSECTION(pch,ints);
xs := ST_X(ST_STARTPOINT(ints));
xe := ST_X(ST_ENDPOINT(ints));
t1 := -d1 - a1*xs; t2 := -d2 - a2*xs;
ys := ((t1*c2)-(t2*c1))/det;
zs := ((b1*t2)-(b2*t1))/det;
t1 := -d1 - a1*xe; t2 := -d2 - a2*xe;
ye := ((t1*c2)-(t2*c1))/det;
ze := ((b1*t2)-(b2*t1))/det;
pxy := ST_FORCE_2D(p2);
gt1 := 'SRID=' || ST_SRID(p2)::text || ';POLYGON(('; gt2 = gt1;
FOR pt IN SELECT geom FROM ST_DUMPPPOINTS(p2) ORDER BY path[1] LOOP
    gt1 = gt1||ST_X(pt)||' '||ST_Z(pt)||',';
    gt2 = gt2||ST_Y(pt)||' '||ST_Z(pt)||',';
END LOOP;
gt1 := substring(gt1 from 1 for length(gt1)-1);
gt2 := substring(gt2 from 1 for length(gt2)-1);
gt1 := gt1 || '))';
gt2 := gt2 || '))';
pxz := ST_GEOMFROMEWKT(gt1);
pyz := ST_GEOMFROMEWKT(gt2);
areaxy := 0; IF ST_ISVALID(pxy) THEN areaxy := ST_AREA(pxy); END IF;
areaxz := 0; IF ST_ISVALID(pxz) THEN areaxz := ST_AREA(pxz); END IF;
areayz := 0; IF ST_ISVALID(pyz) THEN areayz := ST_AREA(pyz); END IF;
IF areaxy > areaxz THEN
    IF areaxy > areayz THEN xs0 := xs;ys0 := ys; xe0 := xe;ye0 := ye;pch := pxy;
    ELSE xs0 := ys;ys0 := zs; xe0 := ye;ye0 := ze;pch := pyz;
    END IF;
ELSE
    IF areaxz > areayz THEN xs0 := xs ys0 := zs; xe0 := xe;ye0 := ze;pch := pxz;
    ELSE xs0 := ys;ys0 := zs; xe0 := ye;ye0 := ze;pch := pyz;
    END IF;
END IF;
ints := ST_GEOMFROMEWKT('SRID='||ST_SRID(p1)||
    ' ;LINESTRING('||xs0||' '||ys0||','||xe0||' '||ye0||')');
IF NOT (ST_CROSSES(pch,ints) OR ST_CONTAINS(pch,ints)) THEN
    RETURN false; END IF;
RETURN true;
END; $$ LANGUAGE plpgsql;
```

SQL function- first part

```
CREATE FUNCTION Ray_3 (geometry, geometry)
  RETURNS integer AS $$
DECLARE ...
BEGIN
p := $1; sup := $2; nint := 0;
x0 := ST_X(p); y0 := ST_Y(p); z0 := ST_Z(p);
FOR pat IN SELECT geom FROM ST_DUMP(sup) LOOP
  i := 1;
  IF (ST_XMAX(pat) >= x0) THEN
    FOR pt IN SELECT geom FROM ST_DUMPPOINTS(patch) LOOP
      IF (i > 3) THEN EXIT; END IF;
      IF (i = 1) THEN x1 := ST_X(pt); y1 := ST_Y(pt); z1 := ST_Z(pt); END IF;
      IF (i = 2) THEN x2 := ST_X(pt); y2 := ST_Y(pt); z2 := ST_Z(pt); END IF;
      IF (i = 3) THEN x3 := ST_X(pt); y3 := ST_Y(pt); z3 := ST_Z(pt); END IF;
      i := i + 1;
    END LOOP;
    a := (y2 - y1)*(z3 - z1)-(z2 - z1)*(y3 - y1);
    b := -((x2 - x1)*(z3 - z1)-(z2 - z1)*(x3 - x1));
    c := (x2 - x1)*(y3 - y1)-(y2 - y1)*(x3 - x1);
    d := - (a*x1 + b*y1 + c*z1);
    IF (a = 0) THEN CONTINUE; END IF;
    xint := -(d + b*y0 + c*z0)/a;
    patxy := ST_FORCE_2D(pat);
    gt1 := 'SRID=' || ST_SRID(pat)::text || ';POLYGON(('; gt2 = gt1;
    FOR pt1 IN SELECT geom FROM ST_DUMPPOINTS(pat) ORDER BY path[1] LOOP
      gt1 = gt1||ST_X(pt1)||' '||ST_Z(pt1)||',';
      gt2 = gt2||ST_Y(pt1)||' '||ST_Z(pt1)||',';
    END LOOP;
    gt1 := substring(gt1 from 1 for length(gt1)-1);
    gt2 := substring(gt2 from 1 for length(gt2)-1);
    gt1 := gt1 || '))'; gt2 := gt2 || '))';
    patxz := ST_GEOMFROMEWKT(gt1);
    patyz := ST_GEOMFROMEWKT(gt2);
    areaxy := 0; IF ST_ISVALID(patxy) THEN areaxy := ST_AREA(patxy); END IF;
    areaxz := 0; IF ST_ISVALID(patxz) THEN areaxz := ST_AREA(patxz); END IF;
    areayz := 0; IF ST_ISVALID(paty) THEN areayz := ST_AREA(paty); END IF;
```

SQL function- second part

```
IF areaxy > areaxz THEN
  IF areaxy > areayz THEN
    p0 := ST_GEOMFROMEWKT('SRID='||ST_SRID(p)||';POINT('||xint||' '||y0||')');
    IF ST_WITHIN(p0,patxy) THEN
      IF (abs(a*x0 + b*y0 + c*z0 + d) <= 10-12) THEN
        RETURN 0;
      END IF;
      nint := nint + 1;
    END IF;
  ELSE
    p0 := ST_GEOMFROMEWKT('SRID='||ST_SRID(p)||';POINT('||y0||' '||z0||')');
    IF ST_WITHIN(p0,patyz) THEN
      IF (abs(a*x0 + b*y0 + c*z0 + d) <= 10-12) THEN
        RETURN 0;
      END IF;
      nint := nint + 1;
    END IF;
  END IF;
ELSE
  IF areaxz > areayz THEN
    p0 := ST_GEOMFROMEWKT('SRID='||ST_SRID(p)||';POINT('||xint||' '||z0||')');
    IF ST_WITHIN(p0,patxz) THEN
      IF (abs(a*x0 + b*y0 + c*z0 + d) <= 10-12) THEN
        RETURN 0;
      END IF;
      nint := nint + 1;
    END IF;
  ELSE
    p0 := ST_GEOMFROMEWKT('SRID='||ST_SRID(p)||';POINT('||y0||' '||z0||')');
    IF ST_WITHIN(p0,patyz) THEN
      IF (abs(a*x0 + b*y0 + c*z0 + d) <= 10-12) THEN
        RETURN 0;
      END IF;
      nint := nint + 1;
    END IF;
  END IF;
END IF;
END LOOP;
RETURN nint;
END; $$ LANGUAGE plpgsql;
```

6 Conclusions and future work

This paper proposed an approach to deal with the problem of specifying spatial integrity constraints at conceptual level in 3D city models written in UML and validating them at physical level (in particular, when data are stored in a spatial DBMS, like PostGIS). The approach is based on OCL templates that allow the model designers to specify semantic properties without using OCL directly. Regarding the implementation of data and their validation procedures, the paper proposed a reference vector model describing the vector types, together with some basic vector operations and predicates that are necessary in order to implement the topological relation tests that are required by the OCL templates. Prop.5.1 showed that using them the topological relation tests used in the examples can be implemented. The proposed set of operations and predicates is not minimal, but this issue is out of the scope of this paper. We prefer to keep some derivable operations and predicates for sake of paper readability. Finally, the feasibility of the implementation on current technology is demonstrated, by choosing PostGIS as representative system of open source spatial databases, and showing the SQL query automatically generated by one of the OCL templates instantiated as an example.

Future work will regard: (i) the extension of the OCL templates to other cases, including also distance based properties in OCL constraint specification; (ii) the implementation of validator tools for city data stored in spatial DBMS; (iii) the testing the approach on huge datasets.

References

- [1] A. Belussi, M. Negri, and G. Pelagatti. An ISO TC 211 Conformant Approach to Model Spatial Integrity Constraints in the Conceptual Design of Geographical Databases. In *Advances in Conceptual Modeling - Theory and Practice*, pages 100–109. 2006.
- [2] A. Belussi, M. Negri, and G. Pelagatti. Modelling Spatial Whole-Part Relationships using an ISO-TC211 Conformant Approach. *Information & Software Technology*, 48(11):1095–1103, 2006.
- [3] B. Demuth, H. Hussmann, and S. Loecher. OCL As a Specification Language for Business Rules in Database Applications. In *Proceedings of the 4th Int. Conf. on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 104–117, 2001.

- [4] M. Duboisset, F. Pinet, M.-A. Kang, and M. Schneider. Precise Modeling and Verification of Topological Integrity Constraints in Spatial Databases: From an Expressive Power study to Code Generation Principles. In *Conceptual Modeling – ER 2005*, pages 465–482. 2005.
- [5] M. J. Egenhofer. Topological relations in 3D. Technical report, National Center for Geographic Information and Analysis, 1995.
- [6] M. J. Egenhofer and R. Franzosa. Point-set topological spatial relations. *International Journal of Geographic Information Systems*, 2(5):161–174, 1991.
- [7] European Commission Joint Research Centre. *INSPIRE Data Specification for the spatial data theme Building*, 2013. version 3.0.
- [8] G. Gröger and L. Plümer. How to Achieve Consistency for 3D City Models. *GeoInformatica*, pages 137–165, 2011.
- [9] G. Gröger and L. Plümer. CityGML – Interoperable Semantic 3D City Models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 71:12–33, 2012.
- [10] J. Herring. *The OpenGIS Abstract Specification, Topic 1: Feature Geometry (ISO 19107 Spatial Schema)*, 2001. Version 5. OGC Document Number 01-101.
- [11] OMG (Object Management Group). *Object Constraint Language (OCL)*, 2014. Version 2.4. OMG Document Number: formal/2014-02-03.
- [12] Open Geospatial Consortium Inc. *OpenGIS Implementation Standard for Geographic Information - Simple feature access - Part 1: Common architecture*, 2011.
- [13] Open Geospatial Consortium Inc. *OGC City Geography Markup Language (CityGML) Encoding Standard*, 2012.
- [14] G. Pelagatti, M. Negri, A. Belussi, and S. Migliorini. From the Conceptual Design of Spatial Constraints to Their Implementation in Real Systems. In *Proc. of the 17th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*, pages 448–451, 2009.
- [15] A. Stadler and T. H. Kolbe. Spatio-Semantic Coherence in the Integration of 3D City Models. In *Proc. of the 5th Int. ISPRS Symposium on Spatial Data Quality (ISSDQ 2007)*, ISPRS Archives, 2007.

- [16] D. Wagner, M. Wewetzer, J. Bogdahn, N. Alam, M. Pries, and V. Coors. Geometric-Semantical Consistency Validation of CityGML Models. In *Progress and New Trends in 3D Geoinformation Sciences*, pages 171–192. 2013.
- [17] D. Xu. Design and Implementation of Constraints for 3D Spatial Database: Using Climate City Campus Database as an Example. Master’s thesis, OTB Research Institute for the Built Environment, 2011.



University of Verona
Department of Computer Science
Strada Le Grazie, 15
I-37134 Verona
Italy

<http://www.di.univr.it>

