GeoInformatica manuscript No.
(will be inserted by the editor)

A Template-based Approach for the Specification of 3D Topological Constraints

Alberto Belussi · Sara Migliorini · Mauro Negri · Giuseppe Pelagatti

Received: date / Accepted: date

Abstract Several different models have been defined in literature for the definition of 3D scenes that include a geometrical representation of objects together with a semantical classification of them. Such semantical characterization encapsulates important details about the object properties and behavior and often includes spatial relations that are defined only implicitly or through natural language, such as "an external access shall be in touch with the building only when it is classified as a direct access". The problem of ensuring the coherence between geometric and semantic information is well known in literature. Many attempts exist which try to extent the OCL to allow the representation of spatial integrity constraints in an UML model. However, this approach requires a deep knowledge of the OCL formalism and the implementation of ad-hoc procedures to validate the constraints specified at conceptual level. Therefore, a new approach is needed that helps designers to define complex OCL constraints and at the same time allows the automatic generation of the code to test them on a given dataset. The aim of this paper is to propose a set of predefined templates to express on the classes of an UML data model, a family of 3D spatial integrity constraints based on topological relations; all this without requiring the knowledge of any formal language by domain experts and supporting their automatic translation into validation procedures.

Keywords 3D topological relations · OCL spatial constraints · spatial constraint specification · spatial conceptual modeling

Alberto Belussi · Sara Migliorni
Department of Computer Science – University of Verona (Italy)
E-mail: sara.migliorini@univr.it

Mauro Negri · Giuseppe Pelagatti
Department of Electronics, Information and Bioengineering – Politecnico of Milan (Italy)

1 Introduction

Many different models exist in literature for representing 3D spatial data and in particular 3D city models, such as CityGML [14], Inspire Annex 3 Building [5] or the Building Theme of the Italian National Core (in Italian, "Edificato"). Besides to the geometrical characterization of objects, which essentially refers to the ISO geometric types [15], such models provide a semantical characterization of objects which encapsulates important meanings and spatial relations. Some examples of them are reported in Sect. 3 in form of integrity constraints.

In general, each data collection C, representing an instance of one of the above cited models, consists of a set of objects $\{o_1, o_2, ...\}$ which has to satisfy two kinds of constraints in order to correctly represent a given scene about a real urban context. The first set of properties, defining the *topological consistency*, regards the correctness of the vector representation (i.e., points, curves, surfaces and solids) that describes the spatial objects $o_1, o_2, ...$ inside the data collection. From a mathematical point of view, these properties derive from the point-set topology and their objective is to ensure the consistency of the data collection as a whole. For this purpose the data collection is represented by means of a topological model (containing primitives like nodes, edges and faces) with a geometrical realization (i.e., nodes as vertices, edges as curve segments and faces as patches). In this way each object o_i is represented by a geometry g_i, which in turn is a set of primitives inside the topological model. The satisfaction of these properties is a necessary condition in order to enable any kind of processing regarding the data collection. The reader can find in [11] an approach for checking *topological consistency* based on sets of axioms defined on a topological model. Since such consistency is invariant with respect to the application context, the approach proposed in [11] can be applied to any data collection, without customization.

The second set of properties, defining the *semantic consistency*, is specified by the domain experts. As a consequence, such properties can change accordingly to the considered application context and they might be applied only to a subset of objects in the collection. They are usually specified at conceptual level by means of a set of *integrity constraints* that frequently includes spatial properties, like topological relations. However, they are often not formally integrated in the model but are implicitly specified or expressed in natural language. Therefore, ad hoc procedures have to be implemented in order to validate and ensure the satisfaction of such constraints. 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 the implementation details and to apply one common constraint framework. In this paper we focus on this second type of properties and on the testing of the semantic consistency.

The Standard ISO TC211 19109 "Rules for application schema" [17] recommends the use of the OCL [22] formalism for specifying spatial integrity constraints at conceptual level. Sect. 2 will discuss several approaches that have been proposed in literature to automatically translate spatially-enhanced

OCL constraints into validation procedures. However, the use of generic OCL constraints has several limitations as deeply discussed in [23]. In particular, it introduces a great complexity both in the conceptual modeling, since a deep knowledge of the OCL language is required by the domain experts, and in the implementation phase, since complex ad-hoc procedures have to be implemented to treat all cases and it is more difficult to optimize such procedures.

This paper follows the approach proposed in [23] that promotes the definition of spatial integrity constraints inside conceptual models through the use of predefined topological constraint templates. Thanks to these templates the designer can specify topological integrity constraints in a straightforward manner, without the need for a deep knowledge of the OCL language. Moreover, since their structure is known, such constraints can be automatically translated into optimized SQL queries or procedures in other programming languages, in order to verify the semantic consistency of any spatial data collection. More specifically, this paper extends the work in [23] to the 3D space, in particular as regards to surfaces and volumes, and it completes the work introduced in [2] through the definition of a wide range of topological templates that potentially capture all common requirements in the definition of a 3D spatial scene. Some testing scenarios are presented in Sect. 3 which originate from the building model of CityGML (LoD3), the INSPIRE Annex 3 Building, and the Italian National Core, but the approach can be easily adapted to other generic 3D spatial models.

The contribution of the paper is presented in Sect. 5 where we illustrate a set of templates that can be instantiated for representing 3D topological constraints at conceptual level with reference to the geometric model described in Sect. 4. These templates can be automatically translated into validation procedures by taking advantages of their mathematical geometric formulation. Finally, some hints about a possible general technique for their translation towards SQL procedures are provided in Sect. 6. The choice to implement the validation procedures as SQL queries is 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 is general enough to be easily implemented using other programming languages or technologies.

2 Related Work

As mentioned in the introduction, two kinds of consistency check can be distiguished: *topological consistency*, which deals with the validation of the vector representation of a set of objects, and semantic consistency, which refers to the checking of spatial integrity constraints based on the semantic characterization of the same set of objects.

Topological consistency – Several works are available in literature which regard the validation of topological consistency in a 3D city model. For instance, in [11] the authors provide a set of axioms to achieve topological con-

sistency of 3D models. The efficiency of the proposed method stems from its locality: in most cases, a dataset can be decomposed into simple components on which basic consistency checks can be performed, then the axioms can be combined to obtain constraints for aggregation of components. With the aim to preserve topological consistency, in [19] and [7] the authors propose the construction of a topological structure that is maintained on top or together with geometric information. Conversely, with reference to the ISO data types, in [18] the author presents a methodology to validate solids against the geometric definition of the ISO Standard 19107.

As regards to CityGML, in [21] the authors define the quality requirements for a general CityGML specification in terms of geometric/topological consistency and provide a suite of essential checking tools to ensure such quality. In [25] the authors define a set of axioms which regards the validation of a dataset based on a CityGML model. In particular, they define the concept of *valid geometry* using the definition of spatial data types provided in [14], and a set of axioms which check the compliance of the dataset w.r.t the types.

Semantic Consistency Given a set of valid geometries, additional *semantic rules* can be defined which are mainly determined by the application context. In [12] the authors provide an overview of CityGML and how it covers the geometrical, topological and semantic aspects of 3D city models. In particular, they highlight how the semantic taxonomy can consistently traverse all the five level of detail (LoD). In [24] the authors distinguish the semantic model of CityGML (e.g., buildings, water bodies, transportation, vegetation) from the geometry model (based on the ISO Standard 19107 [15]) and analyse how to achieve their correspondence. The term *coherence* is used to describe consistent relationships between spatial and semantic entities. Coherence evaluation could be performed by explicitly representing aggregation relations in the semantic model as spatial aggregations, but contrarily to our paper, the authors do not provide any hint about how such constraints can be checked.

Semantic rules are usually defined in terms of OCL [22] statements which refer to the UML class diagram defining the model. The use of OCL [22] for the specification of spatial constraints has been investigated also in [8] 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 alphanumerical and topological constraints. They also investigate the possibility to translate OCL_{9IM} into SQL by providing an extension of the tool named OCL2SQL [6]. In [27] 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 specified in OCL and translated into ad-hoc spatial queries for Oracle. The approach is similar to the one proposed in this paper, however we propose to instantiate constraints by means of templates that can be automatically translated into SQL spatial queries, without any effort for generating new code in each specific case. In [28] the authors present a methodology to model and implement 3D geoconstraints based on four steps: natural language, geometric/topological ab-

stractions, UML/OCL formulations and SQL implementation. However, also in this case constraints have to be defined using complex and ad-hoc OCL formulas, while SQL implementations have to be manually implemented.

The fundamental difference between the template approach proposed in this paper and the generic spatial OCL approach proposed in literature is the following one: instead of augmenting the OCL language with new spatial operators and implementing a generic translator of OCL expressions into a chosen target technology, a set of spatially oriented templates with parameters are defined and an optimized implementation of them is specified for a set of target technologies.

3 Motivating Example

This section illustrates some examples of 3D spatial integrity 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 the Italian National Core and others from CityGML and Inspire, but several other integrity constraints can be defined in other models.

Example 1 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 it must touch it.

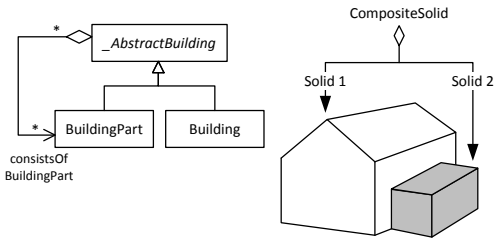

Fig. 1 Example of a building composed of two parts that touch each other.

Example 2 In the conceptual specification of the Italian National Core the following integrity constraint is defined between the UML class representing buildings (Building) and the one representing accesses to buildings (Access): "*an external access shall be in touch with a building only when it is classified as direct*" (see Fig. 2). This means that only the solid representing an access (instance of class Access) having type attribute equal to direct shall be in touch with the solid representing a building (instance of class Building).

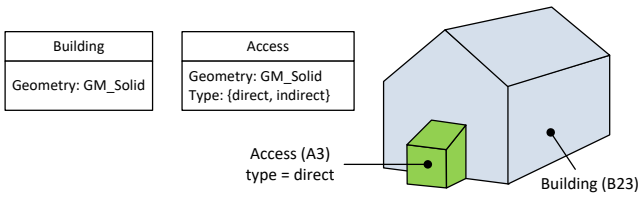

Fig. 2 Example of building and external access that touch each other.

Example 3 With reference to the CityGML and/or the INSPIRE building model, the outer facade of a building can be differentiated semantically using a set of surface types with a special function, like wall, roof, ground, and so on, as illustrated in Fig. 3. Clearly, the following constraint is implicitly defined: "*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*".

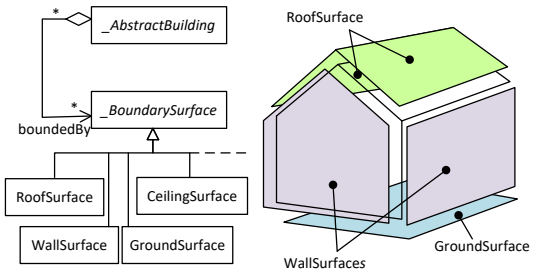

Fig. 3 Example of boundary surfaces of a building.

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).

Example 4 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 lays 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. 4 taken from [14], 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 boundary surface. In this case the integrity constraint is implicitly defined by a conjunction of properties.

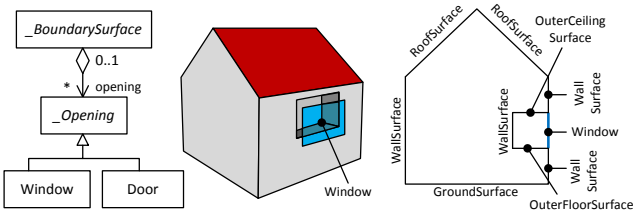

Fig. 4 Example of building with an opening.

4 Geometric Model

This section introduces the geometric model used in the remainder of the paper for the definition of spatial integrity constraints. We will refer to the 3D geometric types described in the ISO Standard 19136 GML [16], which in turn are compliant with the ISO Standard 19107 Spatial Schema [15]. More specifically, this section formalizes the set of considered GML spatial data types and the set of topological relations existing between them, defined in terms of the well-known 9-intersection model [10].

4.1 Spatial Data Types

The 3D spatial data types considered in this paper are taken from the ones formalized in the ISO Standard 19136 GML, since they provide a conformant, partial implementation of the ISO Standard 19107 Spatial Schema.

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 root class representing a type of 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*. These root classes describe the common properties shared by all their subclasses, each of which explicitly prescribe the representation details for generating their instances. Among all general properties, each abstract class provides the concept of *boundary*, *interior* and *exterior* as available methods. These concepts define a partition of the space, in which an object is embedded, producing three point sets that are used to formally specify a reference set of topological relations. The boundary, interior and exterior are formally defined in the point sets topology and in [16] each class exactly provides a specific definition for

them. Intuitively, the boundary separates the interior of an object from the outer space, which represents its exterior.

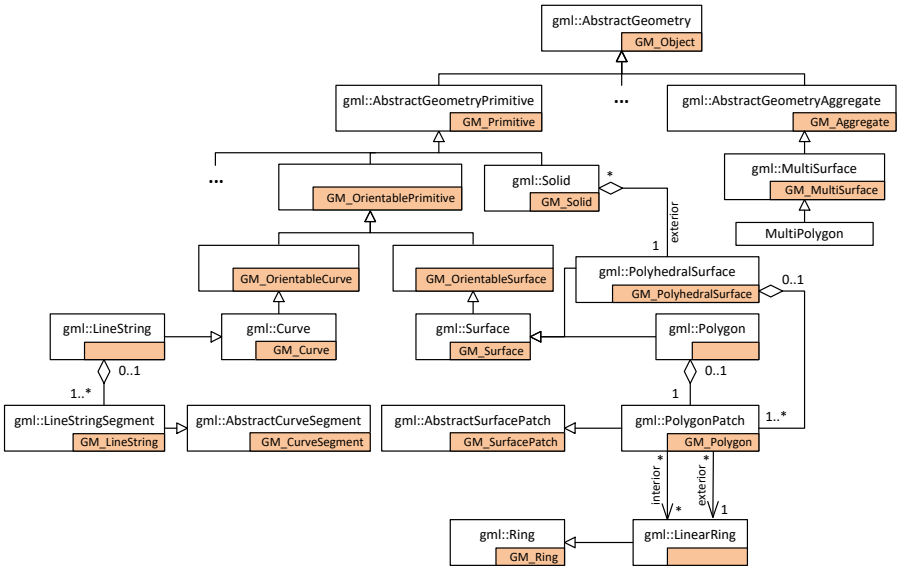

Fig. 5 Hierarchy of spatial data types considered in the paper. The name with the prefix *gml* is the class contained in the ISO Standard 19136 GML, while the name in the small coloured box is the corresponding class in the ISO Standard 19107.

In this paper, we focus on a subset of GML data types which are the most commonly used during the representation of a 3D urban scene. The chosen classes make the approach sufficiently generic without unnecessarily increasing the complexity of the treatment. In particular, the paper considers: (i) line-strings as implementation of curves (ii) polyhedral surfaces and polygons as implementations of surfaces, and (iii) solids defined by means of closed polyhedral surfaces. Fig. 5 shows the hierarchy of considered data types, reporting both the name used in the GML specification (with prefix gml::) and the name of the corresponding class in the ISO Standard 19107 (inside the small coloured inner box). Notice that GML specifies some complementary spatial geometry components, which are not part of the implemented standard (e.g., gml::Polygon and gml::LineString), together with some implementation restrictions which are also valid here.

Definition 1 (Curve) A Curve is a 1-dimensional geometric primitive representing the continuous image of a line. A curve is composed of one or more curve segments which are connected to one another, so that the end point of a segment is the start point of the next one, except for the last one. Each curve segment within a curve represents an homogeneous portion of the curve and may be described using a different interpolation method.

The boundary of a curve is given by the set of points at either end of the curve. In case the two end points coincide, the curve forms a cycle and is said to be closed. The property to be a "continuous image" ensures that the curve is continuous and connected, namely no branchings are allowed inside a curve.

In the ISO Standard 19107, the class GM_Curve is defined as an orientable primitive, namely it carries an orientation property which reflects the orientation in which the curve is traversed. This property is useful when the curve is used as a boundary of an object, since the surface being bounded is the one on the left of the oriented curve. In GML, the orientation of the gml:Curve class is implicitly fixed to positive, and this assumption also holds here.

Each segment of a curve can be defined by a different interpolation method. In this paper, we consider as possible interpolation only the linear one, namely we assume as possible implementation of a curve only the LineString class. Notice that in the ISO Standard 19107, the class GM_LineString is intended as a kind of curve segment, which combines into a single object a sequence of segments, each one consisting in two positions joined by a straight line (i.e., linear interpolation). This class is implemented in GML by the corresponding class gml:LineStringSegment, while the name gml:LineString is used to denote an additional subtype of GM_Curve that consists only of GM_LineString segments. In the following, we use the term LineString in the sense defined in GML.

Definition 2 (LineString) A LineString is a special subtype of GM_Curve that consists of a sequence of segments with linear interpolation. More specifically, each segment inside a linestring is defined by a pair of ordered positions joined by a straight line.

Before discussing the considered 2-dimensional objects, we introduce the concept of LinearRing which will be used for defining the boundary of a polygon. The class gml:LinearRing has been added in GML as a convenience subtype of GM_Ring for denoting a simple ring described by a single closed line-string.

Definition 3 (LinearRing) A LinearRing is a LineString which is closed and simple. A LineString is closed if it is a cycle (i.e., the start and the end vertices coincide) and it is simple if it does not pass through the same point twice with the exception of the two end nodes (i.e., it does not have self-intersection and self-tangency).

A 2-dimensional object is generically represented by the concept of surface. As stated for curves, also the GM_Surface class is considered an orientable primitive in the ISO Standard 19107, while its implementation in GML is given by the class gml:Surface whose orientation is implicitly set to positive. The orientation of a surfaces defines an "up" direction, which, if the surface is not a cycle, is the side of the surface from which the exterior boundary appears counterclockwise.

Definition 4 (Surface) A Surface is a 2-dimensional geometric primitive representing a continuous region of a plane. A surface is defined by one or more

patches, each one representing a homogeneous portion of the plane using a uniform interpolation method. The boundary of a surface is the set of oriented, closed curves (rings) that delineate its limits. The exterior boundary is the one that separates the surface from the infinite space, while the interior boundaries separate the object from other bounded objects.

A surface is said to be *simple* when it consists of a single patch. GML defines a special kind of simple surface which is called Polygon.

Definition 5 (Polygon) A polygon is a special surface that is defined by a *single* surface patch with additional constraints, called PolygonPatch. A PolygonPatch is a surface patch defined by a set of boundary curves that are LinearRings and an underlying surface to which these curves adhere. The curves shall be coplanar and the polygon shall use planar interpolation at its interior.

Notice that the class GM_Polygon defined in the ISO Standard 19107 is a subclass of GM_SurfacePatch which is implemented in GML by the class gml:PolygonPatch. Conversely, the class gml:Polygon has been defined in GML as a convenience subclass of GM_Surface. In the following we will use the term Polygon to denote the concept defined in GML.

This paper considers another surface implementation, admitted by both the ISO Standard 19107 and GML: the PolyhedralSurface, which is also defined using the concept of PolygonPatch and will be used in the definition of solids.

Definition 6 (PolyhedralSurface) A PolyhedralSurface is a surface composed of PolygonPatches connected along their boundary curves. For each pair of patches that touch, the common boundary shall be described as a finite collection of LineStrings. Each of these LineStrings shall be part of the boundary of at most 2 PolygonPatches.

The 3-dimensional geometric type considered in this paper is related to the notion of Solid. In the ISO Standard 19107, the extent of a solid is defined by means of boundary surfaces, called shells, each of which is composed of orientable surfaces connected in a topological cycle. In particular, in a 3D space each solid is limited by one external boundary surface and zero or more internal boundary surfaces. In this paper, we consider a specialization of such generic notion of solid, which is characterized by only one external boundary surface which is a PolyhedralSurface, and zero internal boundaries. It follows that such kind of solid object has no holes (i.e., enclaves). This does not constitute a great limitation since a solid with holes can be replaced by a set of adjacent solids obtained by splitting it [26] into two or more parts. This simplification is required by the model in [9] for the specification of topological relations in 3D. As we will see, it keeps simple the specification of topological relations involving solids without reducing the generality of the model.

Definition 7 (Solid) A Solid is a 3-dimensional object whose extent is defined by a boundary surface, called shell, which is a closed PolyhedralSurface. With reference to the generic definition of solid contained in the standard, this

boundary surface defines the external boundary of the solid, while no internal boundary surfaces are allowed.

Among all possible combinations of geometric primitives provided by the standard, this paper considers only the GM_MultiSurface which is an aggregate class containing only instances of orientable surfaces. An aggregate is an arbitrary collection of geometric objects of a specified type without any additional constraint about its internal structure. Moreover, since this paper considers only a subset of all possible surface types, the multi-surface aggregation is further restricted to be composed only of Polygon instances.

Definition 8 (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.

4.2 Topological Relations

The 9-intersection model [10] is the most common model for defining binary topological relations. It specifies the topological relation R existing between two objects A and B considering the intersection between their interior (A°, B°), boundary (∂A, ∂B) and exterior (A^-, B^-).

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

The topological relations described above apply to primitive types and can be extended to aggregate geometries by imposing some constraints on their components, as formalized in [13] and done in available systems such as PostGIS. For instance, the polygons composing a MultiPolygon (as defined in [13]) cannot overlap. Such constraints do not reduce the expressive power of the type, namely the kind of representable objects, since each generic aggregate can be translated into one that satisfies the given constraints.

In [9] the authors generalize the model to a 3D space. In order to preserve the same properties of 2D relations, in [9] a major assumption is made: the interior, boundary and exterior of volumes are simply connected such that the volume boundary separates the interior from the exterior. Since the volume boundary must be simply connected, the volume cannot have any holes in its interior. This assumption is in accordance with the definition of solid given in Def. 7, where we state that a solid can have only an external boundary but no internal boundaries. Clearly, through a surface external boundary it is always possible to describe solids which are n-tori or donuts. Starting from this model the extension of the definitions to composite/complex solids can be done, by specifying for each relation between two composite/complex solids the

Table 1 3D topological relations between solids (V), surfaces (S), curves (C). Possible topological relations are disjoint (DJ), touch (TC), in (IN), contains (CN), equal (EQ), overlap (OV). The matrix patterns are specified as $1st\ row - 2nd\ row - 3rd\ row$. Used symbols are: T = not empty, F = empty, $*$ = any result, $\mathbf{T}$ = always not empty for the considered combination of geometric types, $T_\odot$ = not empty when the geometries, for which the boundary is considered, are not cycles (e.g., rings are cycles), empty otherwise, T_∂ = not empty, but in the case in which the boundary of the first geometry (e.g., a solid) is equal to the second one (e.g. a surface). Finally, A^T denotes the transpose of a matrix A.

Rel.	Definition	Geom.	Matrix Pattern
DJ	$A \cap B = \emptyset$	V/V	$FFT - FFT - TT\mathbf{T}$
		V/S	$FF\mathbf{T} - FFT - TT_\odot\mathbf{T}$
		V/C	$FF\mathbf{T} - FF\mathbf{T} - TT_\odot\mathbf{T}$
		S/V, C/V	DJ(V/S)^T, DJ(V/C)^T
		S/S, C/C	$FFT - FFT_\odot - TT_\odot\mathbf{T}$
		S/C	$FF\mathbf{T} - FFT_\odot - TT_\odot\mathbf{T}$
		C/S	DJ(S/C)^T
TC	$(A^\circ \cap B^\circ = \emptyset) \wedge (A \cap B \neq \emptyset)$	V/V	$FFT - FTT - TT\mathbf{T}$
		V/S	$FF\mathbf{T} - T*T_\partial - **\mathbf{T} \cup FF\mathbf{T} - FTT - T*\mathbf{T}$
		V/C	$FF\mathbf{T} - T*\mathbf{T} - **\mathbf{T} \cup FF\mathbf{T} - FT\mathbf{T} - T*\mathbf{T}$
		S/V, C/V	TC(V/S)^T, TC(V/C)^T
		S/S, C/C	$FTT - *** - T*\mathbf{T} \cup FFT - T** - T*\mathbf{T} \cup FFT - FT* - T*\mathbf{T}$
		S/C	$FT\mathbf{T} - *** - T*\mathbf{T} \cup FF\mathbf{T} - T** - T*\mathbf{T} \cup FF\mathbf{T} - FT* - T*\mathbf{T}$
		C/S	TC(S/C)^T
IN	$(A \cap B = A) \wedge (A^\circ \cap B^\circ \neq \emptyset)$	V/V	$TFF - T*F - TT\mathbf{T}$
		S/S, C/C	$TFF - **F - TT_\odot\mathbf{T}$
		S/V	$T*F - **F - \mathbf{T}T\mathbf{T}$
		C/V	$T*F - **F - \mathbf{TTT}$
		C/S	$T*F - **F - \mathbf{T}T_\odot\mathbf{T}$
CN	$(A \cap B = B) \wedge (A^\circ \cap B^\circ \neq \emptyset)$	V/V	IN(V/V)^T
		S/S, C/C	IN(S/S)^T, IN(C/C)^T
		V/S	IN(S/V)^T
		V/C	IN(C/V)^T
		S/C	IN(C/S)^T
EQ	$A = B$	V/V, S/S, C/C	$TFF - FTF - FF\mathbf{T}$
OV	$(A^\circ \cap B^\circ \neq \emptyset) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$	V/V	$TTT - TTT - TT\mathbf{T}$
		V/S	$T*\mathbf{T} - T*T_\partial - T*\mathbf{T}$
		V/C	$T*\mathbf{T} - T*\mathbf{T} - T*\mathbf{T}$
		S/S, C/C	$T*T - *** - T*\mathbf{T}$
		S/C	$T*\mathbf{T} - **T_\partial - T*\mathbf{T}$
		S/V, C/V, C/S	OV(V/S)^T, OV(V/C)^T, OV(S/C)^T

corresponding set of expressions that have to be satisfied on their components. However, this is out of the scope of this paper.

As stated in the previous section, the paper concentrates only on 3D data types which are more useful in the description of city models, i.e.: solid (denoted as V), surfaces (denoted as S), which can be Polygons, PolyhedralSurfaces or MultiPolygons, and curves (denoted as C), namely LineStrings. Table 1 re-

ports the formal definition of the topological relations considered in the sequel: they are the relations usually implemented in current GIS systems. For each relation the table shows a name, together with the specification of the pair of geometric types to which it applies, and the corresponding configurations of the 9-intersection matrix representing the scenes where the relation exists; table caption contains more details on the formalism used for representing matrix configurations.

5 Constraint Templates

This section introduces the template-based approach proposed in this paper for the definition of 3D spatial constraints. For each template, we define both its syntax, containing a set of parameters to be specified, and a precise semantics expressed in the OCL language. The translation of each template in the corresponding OCL constraint gives a clear idea about the complexity involved in the specification of the latter with respect to the simplicity of using the former. As discussed in Sect. 2, such spatial constraint templates allow not only to define a constraint in a easy way, but also to obtain optimized implementation for them with respect to a set of target technologies.

The approach includes two main families of constraints: topological and part-whole constraints. The former ones allow to prescribe the existence of a topological relation between the instances of two classes (see Sect. 5.1), while the latter ones allow to describe a composition relation between each instance of a class and a set of instances of the another class (see Sect. 5.2).

5.1 Topological Constraints

A topological constraint between two classes uses the topological relations in Sect. 4.2 for defining conditions on their geometric attributes whose possible types have been specified in Sect. 4.1. A topological constraint template has a fixed logical structure and a set of parameters that allow together to describe the majority of the situations characterizing a 3D city model. In particular, two categories of topological constraints can be identified based on the chosen logical structure: existential topological constraints and universal topological constraints. The first ones prescribe that given an object of the constrained class, there exist at least an object in the constraining class such that the prescribed constraint is satisfied. Conversely, the second ones require that given an object of the constrained class, the prescribed constraint has to be satisfied for each object in the constraining class. For both categories there exists a basic version and several variants each one characterized by an additional set of parameters. In particular, while the basic version of a constraint allows to specify the involved (constrained and constraining) classes, the related spatial attributes, and the required topological relations; a variant may also allow to specify functions to be applied on geometries, or selections on the constrained

and/or the constraining class, or to add an association role between the constrained class and constraining one, that limits the constraint satisfaction to the pairs of instances linked through the given association role. The remainder of this section formalizes all available topological constraint templates by providing both its parametric syntax and the corresponding OCL semantics.

5.1.1 Basic Existential Topological Constraint

The basic existential topological constraint requires that, given an object x belonging to the constrained class X, there exists at least an object y of the constraining class Y such that the given disjunction of topological relations is satisfied between their geometric attributes g and f, given as parameters.

Definition 9 (Basic existential topological constraint ($TC^b_\exists$)) Let X be a constrained class with a spatial attribute g, Y be a constraining class with a spatial attribute f and $\{rel_1|\ldots|rel_k\}$ a disjunction of topological relations. A *basic existential topological constraint* ($TC^b_\exists$) requires that for each object x of X there exists an object y of Y such that one of the relations $rel_1, \ldots, rel_n$ is satisfied between $x.g$ and $y.f$. The semantics of this constraint is represented by means of the following OCL expression.

$TC^b_\exists(X, g, \{rel_1\|\ldots\|rel_k\}, Y, f)$
context X **inv:** Y.`allinstances` $\rightarrow$ `exists`$(a : Y \mid$ `self`$.g$.`check`$(\{rel_1, \ldots, rel_k\}, a.f))$

The OCL statement describing the template semantics is characterized by two parts: the context and the invariant. The context defines the situation to which the constraint is applied and in this case it is represented by the constrained class X. The invariant specifies a condition that must to be true for all instances (i.e., objects) of the context class. Namely in this template, the invariant is evaluated for each object of X and it checks that among all objects of the class Y, there is at least an object such that one of the specified topological relation holds between its geometric attribute f and the attribute g of the currently evaluated object of X. As regards to the constraint syntax:

- `allistances` is an OCL keyword returning all objects of the class on which it is applied. In this case, it returns the set of objects belonging to the constraining class Y.
- `self` is an OCL keyword specifying the **current** instance of the context class which has to be evaluated by the invariant. In this case, it returns the currently considered object of the constrained class X.
- `exists` is an OCL operator on collections of the form $C \rightarrow$ `exists`$(v : T \mid boolExpr(v))$ which checks if for at least one element of the collection C the boolean expression *boolExpr* evaluates to true. The variable v is called iterator, since it is used to iterate among all elements of C, while T denotes the types of such elements.

– `check` verifies that at least one of the topological relation in $\{rel_1, \ldots, rel_k\}$ is satisfied between the two given geometries: $a.\texttt{check}(\{rel_1, \ldots, rel_k\}, b) \equiv_{def} a.rel_1(b) \vee \cdots \vee a.rel_k(b)$. The semantics of $a.rel_i(b)$ has been defined in Tab. 1.

The following example shows how the template $TC^b_\exists$ can be applied to the example in Sect. 3 regarding the relation between a building and its parts.

Example 5 Referring to the Ex. 1 in Sect. 3, 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:

$$TC^b_\exists(\mathsf{BuildingPart}, \mathit{lod3Solid}, \{\mathsf{TC}\}, \mathsf{Building}, \mathit{lod3Solid})$$

Accordingly with CityGML, this example assumes that both classes have a geometric property called *lod3Solid* which represents their extent. Fig. 6 shows a graphical representation of such constraint. In the graphical representation the constraint specification is contained in a comment icon and an arrow represents the direction of the constraint starting from the constrained class and ending onto the constraining class. In order to distinguish this arrow from the one used in UML for depicting an association, it is drawn in blue.

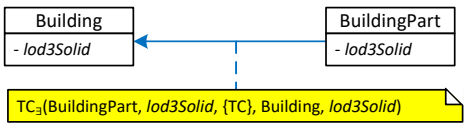

Fig. 6 Example of existential spatial constraint between a main building and its parts.

5.1.2 Existential Topological Constraint with Selection

A variant of the basic existential topological constraint is the one that allows to specify a selection on the instances of the constrained and/or the constraining class, which are considered during the evaluation of the topological constraint.

Definition 10 (Existential topological constraint with selection ($TC^\sigma_\exists$)) Let X be a constrained class with a spatial attribute g, Y be a constraining class with a spatial attribute f and $\{rel_1 \mid \cdots \mid rel_k\}$ a disjunction of topological relations. An *existential topological constraint with selection* ($TC^\sigma_\exists$) allows to express selection conditions for X or Y or both. When a selection σ_1 is specified on X, the constraint is applied only to those instances of X that satisfies σ_1; while, when a selection σ_2 is specified on Y, only those instance of Y that satisfies the selection σ_2 can be considered for testing the required topological relations.

$TC^{\sigma}_{\exists}(X, \sigma_1(X), g, \{rel_1 \mid \ldots \mid rel_k\}, Y, \sigma_2(X,Y), f)$
context X **inv:** $\sigma_1(\texttt{self})$ `implies`(Y.`allinstances` $\rightarrow$ `exists`($a : Y \mid \sigma_2(\texttt{self}, a) \wedge$ (`self`.g.`check`($\{rel_1, \ldots, rel_k\}, a.f$)))

The selection clause $\sigma_1(X)$ is a propositional formula defined as: $\sigma_1 \equiv [\texttt{not}](\alpha_1\ \texttt{lop} \ldots \texttt{lop}\ \alpha_n)$, where α_i is an atomic formula of the form ($X.a$ `op` $X.b$), or ($X.a$ `op cost`), or ($X.a$ `is null`), or ($X.a$ `is not null`), or X.`isTypeOf`(T). The two variables a and b denote attributes of the class X, `op` $\in \{=, >, \geq, <, \leq, \neq\}$ and `const` is a constant value different from `null`. The attributes involved in the formulas can be any attribute of the class X except for the geometric ones. The function `isTypeOf`() is an OCL predefined operation which checks if the current object is an instance of a specified class. This is particularly useful when the current objects belong to a class that is the root of a hierarchy.

The selection clause $\sigma_2(X, Y)$ is a propositional formula similar to $\sigma_1(X)$, but one of the atomic formulas α_i can also be ($X.a$ `op` $Y.c$) or ($X.r_1 = Y.r_2$), namely it can involve attributes of X (a in the formula) and Y (c in the formula) or a role of X (r_1 in the formula) and a role of Y (r_2 in the formula), that can only be compared for equality.

Inside the OCL constraint specification, $\sigma_1(\texttt{self})$ and $\sigma_2(\texttt{self}, a)$ indicate the expressions that are obtained by replacing x with `self` and y with a in $\sigma_1(x)$ and $\sigma_2(x, y)$, respectively. They can be substituted by the value `true` if no selections are required. Since the constraint is applied only to the instances of X that satisfy σ_1, a logical implication is used in the invariant through the OCL operator `implies`. In case the current instance of X satisfies σ_1 (i.e., $\sigma_1(\texttt{self})$=`true`), the existential condition is evaluated by searching, among all instances of Y, the presence of at least one object that satisfies the selection σ_2 and whose geometric attribute is in one of the required topological relations.

The following example shows the application of the template $TC^{\sigma}_{\exists}$ for the formalization of the integrity constraint presented in Sect. 3 that comes from the conceptual specification of the Italian National Core.

Example 6 Referring to Ex. 2 in Sect. 3, an existential topological constraint with selection can be defined between all instances of the class Access that are classified as "direct" and an instance of the class Building. More specifically, given an instance a of Access which is "direct", there exists an instance b of Building such that $a.geometry$ touches $b.geometry$:

$$TC^{\sigma}_{\exists}(\mathsf{Access}, x.\mathsf{type} = \text{“direct”}, \mathit{geometry}, \{\mathsf{TC}\}, \mathsf{Building}, \texttt{true}, \mathit{geometry})$$

This example assumes that both classes have a geometric property called *geometry* which represents their 3D representation. Fig. 7 shows a graphical representation of such constraint.

The example below includes some selection conditions in order to represent the relation between an opening and the surface that contains it.

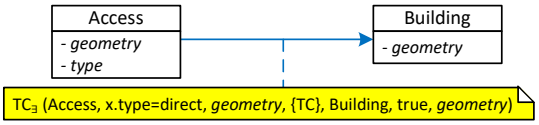

Fig. 7 Example of existential topological constraint between a direct access and its building.

Example 7 With reference to the CityGML data model, we require that the surface of an _Opening (*lod3MultiSurface*) shall be contained into a _BoundarySurface (see Fig. 8). However, not all kinds of boundary surface can contain a window or a door: it is reasonable to assume that only an instance of RoofSurface or of WallSurface can contain a window or a door. This requirement can be expressed by means of an integrity constraint using a selection on the surface type as follows.

$$TC^{\sigma}_{\exists}(\text{\_Opening}, \texttt{true}, \textit{lod3MultiSurface}, \{\text{IN}\}, \text{\_BoundarySurface}, y.\texttt{IsTypeOf}(\text{RoofSurface}, \text{WallSurface}), \textit{lod3MultiSurface})$$

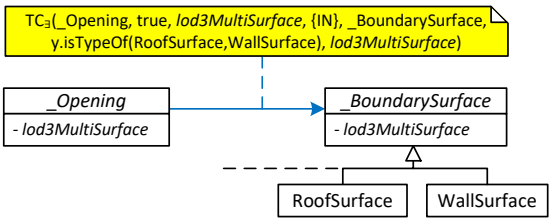

Fig. 8 Example of existential topological constraint with selection between an opening and the boundary surface of its building.

5.1.3 Existential Topological Constraint with Spatial Function

This variant of the basic existential topological constraint allows to specify some spatial functions that have to be applied on the geometric attribute of the constrained, or the constraining class, or both. In this way, the constraint is verified on the geometric value produced by the spatial function, instead of on the geometric attribute itself. Examples of spatial functions commonly used in the specification of topological constraints, are the ones that return the boundary or the planar projection of a geometry.

Definition 11 (Existential topological constraint with spatial function ($TC^{s}_{\exists}$)) Let X be a constrained class with a spatial attribute g, Y be a

constraining class with a spatial attribute f and $rel_1, \ldots, rel_n$ a disjunction of topological relations. An *existential topological constraint with spatial function* ($TC^s_\exists$) allows to apply a spatial function $s_1()$ on the geometric attribute g of X, and/or $s_2()$ on the geometric attribute f of Y, in order to obtain the spatial values on which the topological relations have to be evaluated.

$TC^s_\exists(X, g, s_1(), \{rel_1 \mid \ldots \mid rel_n\}, Y, f, s_2())$
context X **inv:** Y.`allinstances` $\rightarrow$ `exists`$(a : Y \mid$ `self`$.g.s_1()$.`check`$(\{rel_1, \ldots, rel_k\}, a.f.s_2()))$

Functions $s_1()$ and $s_2()$ are any spatial function that can be applied to the geometric attribute of X and Y, respectively. If $s_1()$ (or $s_2()$) is `null` in the template, then the function is not inserted in the OCL invariant, i.e. $g.s_1()$ is equal to g (or $f.s_2()$ is equal to f). □

The following example makes use of a spatial function for constraining the geometry of an opening to the building solid.

Example 8 The existential topological constraint using the function *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 example of data model presented in Fig. 9, which comes from CityGML, windows are represented as surfaces having their boundary that lies completely inside the boundary of the solid representing a building. Therefore, it is necessary to ensure that such surface boundary is contained in the solid boundary of a building by means of the following constraint.

$$TC^s_\exists(\mathsf{Window}, \mathit{lod3MultiSurface}, \mathit{boundary}(), \{\mathsf{IN}\}, \mathsf{Building}, \mathit{lod3Solid}, \mathit{boundary}())$$

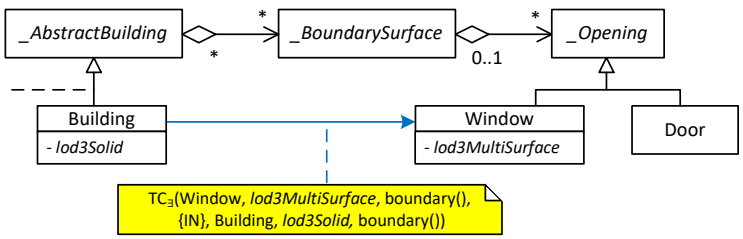

Fig. 9 Example of existential topological constraint with application of a spatial function on the geometric attribute.

5.1.4 Generic Existential Topological Constraint

Clearly, the two variants presented before can be combined obtaining a generic existential topological constraint with the following form, where unused selections can be replaced by the value `true`, and unused spatial functions by the value `null`.

$TC^{*}_{\exists}(X, \sigma_1(x), g, s_1(), \{rel_1 \vert .. \vert rel_n\}, Y, \sigma_2(x,y), f, s_2())$
context X **inv:** $\sigma_1(\texttt{self})$ `implies`$(Y.\texttt{allinstances} \rightarrow$ `exists`$(a : Y \mid \sigma_2(\texttt{self}, a) \wedge$ `self`$.g.s_1().\texttt{check}(\{rel_1, \ldots, rel_n\}, a.f.s_2())))$

5.1.5 Existential Topological Constraint based on a Chain of Roles

In some cases it is necessary to specify a spatial constraint based on an association that links the constrained class with the constraining one. The existential constraint presented in this section allows to consider as available instances of the constraining class, only the objects that can be reached from the constrained object through a chain of roles $r_1, \ldots r_n$.

Definition 12 (Existential topological constraint with roles ($TC^{r}_{\exists}$)) Let X be a constrained class with a spatial attribute g, Y be a constraining class with a spatial attribute f and $\{rel_1 | \ldots | rel_n\}$ a disjunction of topological relations. An *existential topological constraint with roles* ($TC^{r}_{\exists}$) allows to specify a chain of roles $r_1, \ldots, r_n$, so that the available objects of the constraining class are only those that can be reached from the constrained object through such chain.

$TC^{r}_{\exists}(X, \sigma_1(X), (r_1, \ldots, r_n), g, s_1(), \{rel_1 \vert \ldots \vert rel_n\}, Y, \sigma_2(X,Y), f, s_2())$
context X **inv:** $\sigma_1(\texttt{self})$ `implies` $(\texttt{self}.r_1 \rightarrow$ `collect`$(b_1 \mid b_1.r_2) \rightarrow \cdots \rightarrow$ `collect`$(b_{n-1} \mid b_{n-1}.r_n) \rightarrow$ `exists`$(a : Y \mid \sigma_2(\texttt{self}, a) \wedge$ `self`$.g.s_1().\texttt{check}(\{rel_1, \ldots, rel_n\}, a.f.s_2())))$

Notice that the $TC^{r}_{\exists}$ constraint is defined starting from the generic existential topological constraint in order to provide a maximum degree of flexibility in its definition and in particular to allow its combination with all the variants described before. The main difference between $TC^{*}_{\exists}$ and $TC^{r}_{\exists}$ is in the collection of objects of the constraining class: while in the former it is represented by all instances of the class Y, in the latter it is obtained starting from the current object of X and traversing the chain of links. The `collect` is an OCL operator on collections of the form $C \rightarrow \texttt{collect}(v : T \mid expr(v))$ which starting from the available collection C derives a different collection containing the objects returned by the expression *expr* evaluated on the objects of C. In the invariant, the statement $\texttt{self}.r_1 \rightarrow \texttt{collect}(b_1 \mid b_1.r_2)$ starts from the collection C of objects obtained by navigating the role r_1 from the current

object of X ($C = \texttt{self}.r_1$), and produces a new collection containing all the objects reachable by applying the role r_2 to the objects in C. Supposing that only two roles are specified in the chain, then the class C is the intermediate class used in the navigation for reaching the objects of the class Y starting from the object of X.

Example 9 The existential spatial constraint based on a chain of roles can be used to model an additional application-specific constraint regarding the BuildingInstallation class of CityGML (see Fig. 10), which would require that the objects of this class that are classified as *chimney* (class=*'chimney'*) must have a projection in 2D that is contained in at least one of the `_BoundarySurfaces` of the `Building` it belongs to and that the `_BoundarySurface` is instance of the class `GroundSurface`.

$$
\begin{aligned}
&TC^{r}_{\exists}(\mathsf{BuildingInstallation}, \mathsf{class} = \text{'chimney'},\\
&\quad (\mathsf{\_abstractBuilding}, \mathsf{boundedBy}),\\
&\quad \mathit{lod3Geometry}, \mathit{planarProjection}(), \{\mathsf{IN}\},\\
&\quad \mathsf{\_BoundarySurface}, \mathtt{isTypeOf}(\mathsf{GroundSurface}),\\
&\quad \mathit{lod3MultiSurface}, \mathit{planarProjection}())
\end{aligned}
$$

In the example, the function *planarProjection*() is used. It simply drops the third coordinate from the input geometry g. The role _abstractBuilding is used for navigating the association that links BuildingInstallation to _AbstractBuilding. It is the OCL syntax for the navigation of association without role names.

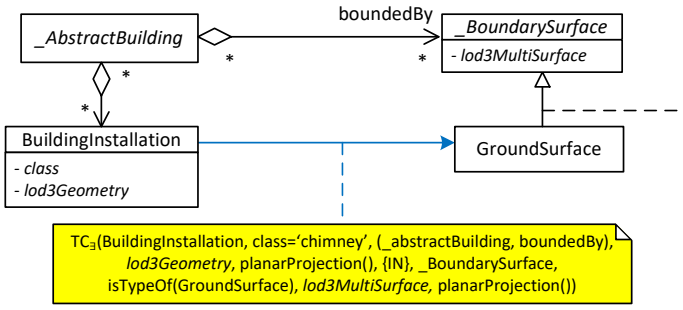

Fig. 10 Example of existential topological constraint with a chain of roles between an installation on a building and the boundary surfaces of its building.

5.1.6 Basic Universal Topological Constraint

The second kind of considered logical structure is the universal one which replaces the existential quantification with a universal one, similarly to what have been proposed in [4]. In this kind of constraints it is required that given an object x of the constrained class X, the constraint has to be satisfied for all object y of the constraining class Y.

Definition 13 (Basic universal topological constraint ($TC^b_\forall$)) Let X be a constrained class with a geometric attribute g, Y be a constraining class with a geometric attribute f and $\{rel_1 \mid \cdots \mid rel_n\}$ a disjunction of topological relations. A *basic universal topological constraint* ($TC^b_\forall$) requires that one of the topological relations $rel_1, \ldots, rel_n$ 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.

$TC^b_\forall(X, g, \{rel_1 \mid \ldots \mid rel_n\}, Y, f)$
context X **inv:** Y.`allinstances` $\rightarrow$ `forall`$(a : Y \mid$ `self`$.g.($`check`$(\{rel_1, \ldots, rel_k\}, a.f))$

The OCL invariant describing $TC^b_\forall$ is similar to the one defined for the basic existential topological constraint in Sect. 5.1.1. The only difference regards the use of the collection operator `forall` in place of the corresponding operator `exists`. As the name suggests, this operator checks if the specified expression holds for all elements of the collection to which it has been applied.

The $TC^b_\forall$ constraint is meaningful only for some kinds of topological relations, for instance *disjoint* or *touch*, as illustrated in the following examples.

Example 10 A typical usage for the universal spatial constraint is for establishing that every building must be disjoint from or be in touch with each other building:

$$TC^b_\forall(\texttt{Building}, \textit{lod3Solid}, \{\texttt{TC}, \texttt{DJ}\}, \texttt{Building}, \textit{lod3Solid})$$

Clearly, when the same class is used both as constrained and constraining class, the test is performed by considering for each object of the class, all the other objects of the same class excluding itself.

5.1.7 Universal Topological Constraint with Selection

Selection conditions can be applied on both the constraining and the constrained class with the same meaning and considerations made for the corresponding existential variant presented in Sect. 5.1.2.

Definition 14 (Universal topological constraint with selection ($TC^\sigma_\forall$)) Let X be a constrained class with a geometric attribute g, Y be a constraining

class with a geometric attribute f and $\{rel_1 \mid \cdots \mid rel_n\}$ a disjunction of topological relations. A *universal topological constraint with selection* ($TC^{\sigma}_{\forall}$) allows one to express selection conditions for X or Y or both. When a selection σ_1 is specified on X, the constraint is applied only to those instances of X that satisfies σ_1; while, when a selection σ_2 is specified on Y, only those instance of Y that satisfies the selection σ_2 can be considered for testing the required topological relations.

$TC^{\sigma}_{\forall}(X, \sigma_1(X), g, \{rel_1 \mid \ldots \mid rel_n\}, Y, \sigma_2(X,Y), f)$
context X **inv:** $\sigma_1(\texttt{self})$ `implies` $(Y.\texttt{allinstances} \rightarrow$ $\texttt{select}(a : Y \mid \sigma_2(\texttt{self}, a)) \rightarrow$ $\texttt{forAll}(b : Y \mid \texttt{self}.g.\texttt{check}(\{rel_1, \ldots, rel_k\}, a.f)))$

In the case of a universal constraint, the selection condition σ_2 applied to the constraining class Y cannot be placed in conjunction with the check of the topological relations. Conversely, it is required to extract from the set of all objects of Y only those that satisfy the selection and then check only on them the topological relations. For this reason, the `select` operator has been applied in order to retrieve the considered set of constraining objects. It has the form $C \rightarrow \texttt{select}(v : T \mid boolExpr(v))$ and produces a subset of C containing only the elements that satisfy the defined boolean expression.

Example 11 With reference to the CityGML data model (see Fig. 11), we can specify by using the following additional constraint that an object representing an instance of Window (subclass of _Opening) must be disjoint from any surface representing an object of both the GroundSurface class and the OuterCeilingSurface class:

$$TC^{\sigma}_{\forall}(\text{Window}, \texttt{true}, \textit{lod3MultiSurface}, \{\text{DJ}\}, \text{\_BoundarySurface},$$
$$(y.\texttt{isTypeOf}(\text{GroundSurface}) \vee y.\texttt{isTypeOf}(\text{OuterCeilingSurface})),$$
$$\textit{lod3MultiSurface})$$

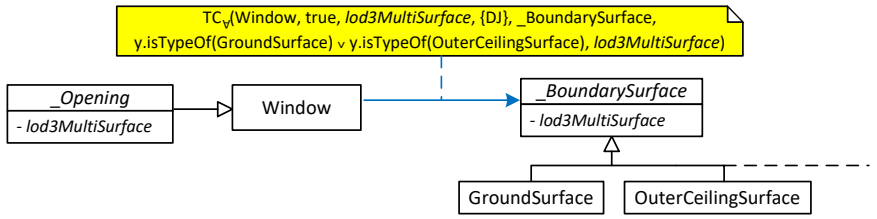

Fig. 11 Example of universal topological constraint with a selection between a window opening and the boundary surfaces of kind ground or outer ceiling.

5.1.8 Universal Topological Constraint with Spatial Functions

Spatial functions can be applied to the geometric attribute of both the constraining and the constrained class with the same meaning and considerations made for the corresponding existential variant presented in Sect. 5.1.3.

Definition 15 (Universal topological constraint with spatial functions ($TC^s_\forall$)) Let X be a constrained class with a geometric attribute g, Y be a constraining class with a geometric attribute f and $\{rel_1|\ldots|\{rel_n\}$ a disjunction of relations. A *universal topological constraint with spatial functions* ($TC^s_\forall$) allows to apply a spatial function $s_1()$ on the geometric attribute g of X, and/or a spatial function $s_2()$ on the geometric attribute f of Y.

$TC^s_\forall(X, g, s_1(), \{rel_1|\ldots|rel_n\}, Y, f, s_2())$
context X **inv:** Y.`allinstances` $\rightarrow$ `forall`$(a : Y \mid$ `self`$.g.s_1()$.`check`$(\{rel_1, \ldots, rel_k\}, a.f.s_2()))$

Examples of this type of constraint can be obtained every time it is necessary to specify a condition on the boundary or on the planar projection of a geometric attribute of a class in a universal topological constraint.

5.1.9 Generic Universal Topological Constraint

As done in Sect. 5.1.4 for the existential topological constraint, it is possible to generalize also the universal one by combining the two variants previously described which allow the definition of both selections on the constrained and constraining objects, and spatial functions on the involved geometric attributes. Also in this case, unused selection conditions can be replaced by the value `true`, while unused spatial functions with the value `null`.

$TC^*_\forall(X, \sigma_1(x), g, s_1(), \{rel_1|..|rel_n\}, Y, \sigma_2(x, y), f, s_2())$
context X **inv:** σ_1(`self`) `implies`(Y.`allinstances` $\rightarrow$ `select`$(a : Y \mid \sigma_2($`self`$, a)) \rightarrow$ `self`$.g.s_1()$.`check`$(\{rel_1, \ldots, rel_k\}, a.f.s_2()))$

5.1.10 Universal Topological Constraint based on a Chain of Roles

The final kind of universal topological constraint proposed in this paper is the one that allows to specify a chain of $r_1 \ldots r_n$ roles.

Definition 16 (Universal topological constraint with roles ($TC^r_\forall$)) Let X be a constrained class with a geometric attribute g, Y be a constraining class with a geometric attribute f, $\{rel_1|\ldots|rel_n\}$ be a disjunction of topological relations, and $r_1, \ldots, r_n$ be a chain of association roles that links the class X to the class Y. A *universal topological constraint with roles* ($TC^r_\forall$) considers as available objects of the constraining class, only the objects that can be reached from the constrained object through the chain of roles $r_1 \ldots . r_n$.

$TC^r_\forall(X, \sigma_1(X), (r_1, \ldots, r_n), g, s_1(), \{rel_1 \mid \ldots \mid rel_n\}, Y, \sigma_2(X, Y), f, s_2())$
context X **inv:** $\sigma_1(\texttt{self})$ `implies` $(\texttt{self}.r_1 \rightarrow$ $\texttt{collect}(b_2 \mid b_2.r_2) \rightarrow \cdots \rightarrow \texttt{collect}(b_n \mid b_n.r_n) \rightarrow$ $\texttt{select}(a : Y \mid \sigma_2(\texttt{self}, a)) \rightarrow$ $\texttt{forall}(b : Y \mid \texttt{self}.g.s_1().\texttt{check}(\{rel_1, \ldots, rel_n\}, b.f.s_2())))$

The OCL invariant uses a construction similar to the one applied in Sect. 5.1.5 to retrieve the set of constraining objects starting from the chain of roles. Given such set of retrieved constrained objects, an eventual selection is applied on them and finally the topological check is performed.

This template can be applied to model some conditions regarding the openings in CityGML.

Example 12 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. 4. This condition can be represented through this pair of constraints:

$$TC^r_\forall(\mathsf{Building}, \texttt{true}, (\mathsf{boundedBy}, \mathsf{opening}), \mathit{lod3Solid}, \mathit{boundary}(), \{\mathsf{CN}\}, \mathsf{\_Opening}, \texttt{true}, \mathit{lod3MultiSurface}, \mathit{boundary}())$$

$$TC^r_\forall(\mathsf{Building}, \texttt{true}, (\mathsf{boundedBy}, \mathsf{opening}), \mathit{lod3Solid}, \mathit{boundary}(), \{\mathsf{TC}\}, \mathsf{\_Opening}, \texttt{true}, \mathit{lod3MultiSurface}, \texttt{null})$$

The second constraints ensures that each instance of _Opening does not intersect the solid representing the building volume.

5.2 Part-whole Constraint

The Part-whole constraints are a family of integrity constraints that involve two classes of the schema where the first class C_{whole} represents the instances of the composed objects and the second one C_{part} represents the instances of the composing objects [3]. Usually an association between them A_{pw} (with roles: whole and parts) is used for denoting the group of parts $p \in C_{\mathsf{part}}$ that compose a given whole $w \in C_{\mathsf{whole}}$.

The Part-whole constraint is usually composed of two sections: (i) the first one requires the satisfaction of a condition by the parts with respect to the whole; (ii) the second one constraints the whole with respect to the parts. The first constraint can be specified by using one of the previous template, by prescribing for example that the geometry g of each part must be spatially contained into the geometry f of one whole:

$$TC^b_\exists(\mathsf{C_{part}}, g, \{\mathsf{IN}\}, \mathsf{C_{whole}}, f)$$

or, when the association A_{pw} exists the template based on the chain of roles can be used:

$$TC^{r}_{\forall}(\mathsf{C_{part}}, \texttt{true}, (\mathsf{whole}), g, \texttt{null}, \{\mathsf{IN}\}, \mathsf{C_{whole}}, \texttt{true}, f, \texttt{null})$$

Therefore, no additional templates are necessary for the first part of the Part-whole constraint. On the other hand, the second part cannot be expressed by using the previously defined templates, since it usually requires the specification of a spatial predicate that involves one *whole* and a set of parts *parts*, requiring that the spatial union of the *parts* is spatially equal to the *whole*. Thus, a new template has to be defined.

Definition 17 (Composed-of constraint ($PW_{\uplus}$)) Let C_{whole} be a constrained class with a geometric attribute g, C_{part} be a constraining class with a geometric attribute f. A *composed-of constraint* ($PW_{\uplus}$) requires that for each $x \in C_{\mathsf{whole}}$ there must exist a set of *parts* $P \subseteq C_{\mathsf{parts}}$, such that the spatial union of the parts is spatially equal to the whole.

$PW_{\uplus}(C_{\mathsf{whole}}, g, C_{\mathsf{parts}}, f)$
context $\mathsf{C_{whole}}$ **inv:** $\texttt{self}.g.\texttt{check}(\{\mathsf{EQ}\},\ \mathsf{C_{parts}}.\texttt{allInstances}.\texttt{f} \rightarrow$ $\quad \texttt{select}(a : \mathsf{GM\_Object} \mid \texttt{self}.g.\texttt{check}(\{\mathsf{CN}, \mathsf{EQ}\}, a)) \rightarrow$ $\quad \texttt{iterate}(b : \mathsf{GM\_Object}, acc : \mathsf{GM\_Object} = \emptyset \mid acc.\texttt{gUnion}(b)))$

The function $a.\texttt{gUnion}(b)$ computes the geometric 3D union of the geometries a and b.

Example 13 This template can be applied to specify some requirements of the Italian National Core regarding the relationships between the class Building and the class VolumetricUnit. A building is subdivided into volumetric units, each one characterized by a uniform eaves height (z-value). In particular, we can express by a constraint $PW_{\uplus}$ that the solid of each building is always the composition of a set of solids representing instances of the class VolumetricUnit (see Fig. 12).

$$PW_{\uplus}(\mathsf{Building}, \mathit{geometry}, \mathsf{VolumetricUnit}, \mathit{geometry})$$

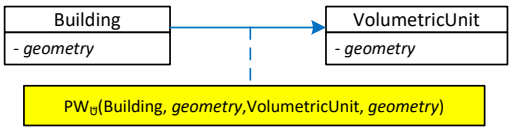

Fig. 12 Example of part-whole constraint between a building and a set of volumetric units.

By exploiting the association A_{pw}, the following variant of the $PW_{\uplus}$ template can be introduced, where instead of selecting the parts by testing whether their geometry is contained inside the geometry of the whole, the association is used to link a whole to its parts.

Definition 18 (Composed-of constraint based on the association ($PW^r_{\uplus}$)) Let C_{whole} be a constrained class with a geometric attribute g, C_{part} be a constraining class with a geometric attribute f, and A_{pw} be an association linking each whole with its parts through the role *parts*. A *composed of constraint based on the association* A_{pw} ($PW^r_{\uplus}$) requires that for each $w \in C_{\mathsf{whole}}$ the set of parts that can be obtained by navigating the role $w.parts$ must satisfy the following property: the spatial union of the set *parts* shall be spatially equal to the whole w.

$PW^r_{\uplus}(C_{\mathsf{whole}}, parts, g, C_{\mathsf{parts}}, f)$
context $\mathsf{C}_{\mathsf{whole}}$ **inv:** $\texttt{self}.g.\texttt{check}(\{EQ\}, \texttt{self}.parts.f \rightarrow$ $\quad \texttt{iterate}(b : \mathsf{GM\_Object}, acc : \mathsf{GM\_Object} = \emptyset \mid acc.\texttt{gUnion}(b)))$

Similarly to what have been done for the topological constraints, from the $PW_{\uplus}$ template we can obtain the variants with selections and spatial functions producing the template $PW^{\sigma}_{\uplus}$ and $PW^{s}_{\uplus}$, respectively.

In the following section the problem of testing the satisfaction of the set of constraints defined on a given UML data model is discussed. In particular, we give some hints about a possible implementation of the templates in SQL (the chosen platform is PostgreSQL with PostGIS). This is one of the possible implementations, other technologies can be applied for the constraints validation, for instance systems like SpatialHadoop of the big data family could be used instead of PostgreSQL [20].

6 SQL Implementation of Templates

This section gives some hints about the possibility to automatically translate the constraint templates presented in Sect. 5 into SQL executable procedures. The general idea is that such SQL procedures can be instantiated and executed on a relational database implementing the UML classes through a set of relational tables. The relational schema can been obtained by following some mapping rules that guarantee the ability to represent all possible states of the objects describing an instance of the given UML classes. For example, each constrained class X (constraining class Y) is represented by a table $\texttt{T\_}X$ ($\texttt{T\_}Y$) containing all its non multi-value properties as columns with the same name. Conversely, multi-value properties and $n{-}n$ associations are represented through additional tables. Notice that X and Y are generic class names that will be actually instantiated with specific class names, when corresponding tables are created.

The remainder of this section shows the implementation for the templates $TC^{*}_{\exists}$, $TC^{*}_{\forall}$, $TC^{r}_{\forall}$ and $PW_{\uplus}$ by providing for each of them the corresponding SQL query that is able to extract all the tuples containing objects of the constrained class that violate the constraint. The implementations for the other templates can be easily derived from these representative examples. In the SQL queries, parameters are written in italics: X and Y generically denote

the class names, while the tuple variables `x` and `y`) are used to refer to a tuple of the constrained table `T_`X or constraining table `T_`Y, respectively. F_{s_1} and F_{s_2} are the PostGIS functions corresponding to $s_1()$ and $s_2()$, respectively, the function $F_{\cup}$ represents the implementation in PostGIS of the union of two geometries in 3D. Each function $F_{rel_i}(g, f)$ denotes the test required to check the existence of the topological relation rel_i between g and f in 3D. Currently available GIS systems usually provide support for testing 2D topological relations, while the support for the corresponding 3D functions is quite limited. Anyway, starting from the formal definition of topological relations given in Sect. 4.2, it is possible to reduce such tests to a set of tests performed on elementary object components (such as the boundary, the planar interior parts, and so on). It follows that these 3D functions can be implemented by combining a set of basic operations commonly available in GIS systems, such as `ST_DumpPoints()`, `ST_Boundary()`, `ST_3DIntersection()` and the various 2D topological relations provided by PostGIS, together with a limited set of ad-hoc custom implementations. Notice that such ad-hoc implementations have to be done only once for all constraint templates and they can be replaced by standard implementations as soon as they become available. As a future work, we will prove the feasibility of such derivation.

SQL implementation for $TC^*_{\exists}$

$TC^*_{\exists}(X, \sigma_1(x), g, s_1(), \{\texttt{rel}_1|\ldots|\texttt{rel}_n\}, Y, \sigma_2(x, y), f, s_2())$

SQL query

```
SELECT x.* FROM T_X as x
WHERE σ1(x) AND
  NOT EXISTS( SELECT 1 FROM T_Y as y
              WHERE σ2(x,y) AND
                    (Frel1(Fs1(x.g), Fs2(y.f)) OR...OR
                     Frel_n(Fs1(x.g), Fs2(y.f))))
```

SQL implementation for $TC^*_{\forall}$

$TC^*_{\forall}(X, \sigma_1(x), g, s_1(), \{rel_1|..|rel_n\}, Y, \sigma_2(x, y), f, s_2())$

SQL query

```
SELECT x.* FROM T_X as x
WHERE σ1(x) AND
   EXISTS( SELECT 1 FROM T_Y as y
           WHERE σ2(x,y) AND
                 NOT(Frel1(Fs1(x.g), Fs2(y.f)) OR...OR
                     Frel_n(Fs1(x.g), Fs2(y.f))))
```

SQL implementation for $TC^r_\forall$

$TC^r_\forall(X, \sigma_1(x), (r_1, .., r_m), g, s_1(), \{rel_1|..|rel_n\}, Y, \sigma_2(x, y), f, s_2())$

SQL query

```
SELECT x.* FROM T_X as x
WHERE σ1(x) AND EXISTS
       (SELECT 1 FROM T_Y as y
          JOIN T_{m-1} as y_{m-1} ON y_{m-1}.r_m = y.ID
          JOIN ... JOIN T_1 as y_1 ON y_1.r_2 = y_2.ID
        WHERE x.r_1 = y_1.ID AND σ2(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))))
```

SQL implementation for $PW_\uplus$

$PW_\uplus(C_{\mathsf{whole}}, g, C_{\mathsf{part}}, f)$

SQL query

```
SELECT x.* FROM T_C_whole as x, T_C_part as y
WHERE F_CN(x.g, y.f) OR F_EQ(x.g, y.f)
GROUP BY x.ID
HAVING NOT F_EQ(x.g, F_∪(y.f))
```

For example, the query for testing the first constraint of Ex. 12 can be obtained by instantiating the SQL query $TC^r_\forall$, obtaining the following:

SQL Query

```
SELECT x.*
FROM T_Building as x
WHERE true AND EXISTS
  (SELECT 1
   FROM V_Opening as y JOIN V_BoundarySurface as y1
     ON y1.opening=y.ID
   WHERE x.boundedBy = y1.ID AND true AND
     NOT(F_CN(ST_Boundary(x.lod3Solid),
              ST_Boundary(y.lod3MultiSurface)))
```

7 Conclusions and future work

This paper proposes an approach to deal with the problem of specifying spatial integrity constraints at conceptual level in 3D city models written in UML. In particular, the approach uses a set of predefined constraint templates, which allow the model designers to specify semantic properties without using OCL directly. The use of such templates not only eases the modeling activity, but also avoids the use of custom ad-hoc implementation for each single constraint, opening the way to more optimized validation procedures.

The proposed topological constraint templates have been defined with reference to a set of 3D geometric types compliant with the ISO Standard 19107, and a set of 3D topological relations defined by means of the well-known

9-intersection model. Each template is characterized by a logical structure (existential or universal) and a topological structure given by a disjunction of topological relations that have to be checked between pair of geometries. Some extensions of the basic template forms have been provided which include the possibility to define selections on both the constrained and the constraining classes, apply spatial functions on the geometric attributes, or use a chain of association roles in order to identify the desired constraining instances.

For each defined template, the paper provides its formalization in OCL, giving a clear idea about the different level of complexity between the use of the templates and of the direct use of the OCL language. Moreover, some examples are described which refers to CityGML, Inspire and the Italian National Core.

Finally, the paper provides some hints about the feasibility of the implementation of these templates using the current technology, by choosing PostGIS as representative system. In particular, it shows an example of SQL query that can be automatically generated by one instantiated OCL template.

Regarding to the expressiveness of the proposed framework, we can observe that the set of the defined templates does not cover the set of all possible OCL expressions that can be specified on a given UML class diagram. However, although the proposed framework contains only six templates: (i) two template for expressing existential topological constraints ($TC^{*}_{\exists}$ and $TC^{r}_{\exists}$); (ii) two template for expressing universal topological constraints ($TC^{*}_{\forall}$ and $TC^{r}_{\forall}$); (iii) two additional template for expressing part-whole constraints ($PW^{*}_{\uplus}$ and $PW^{r}_{\uplus}$); it is able to specify the most interesting constraints of the considered UML class diagrams (i.e., CityGML, Inspire and the Italian National Core). In particular, we can express:

- all the possible existential and universal constraints testing a disjunction of the topological relations in Tab. 1 involving two geometric attributes g and f of two classes X and Y of the given class diagram;
- all the variants of this first set of constraints that can be obtained by selecting the constrained objects and the constraining objects by means of predicates on their local properties (attributes and roles) and/or predicates that specify join conditions between X and Y;
- all the variants produced by introducing geometric functions that can modify the geometries before testing the topological relations;
- all the variants produced by pruning the constraining object by means of a chain of association roles;
- all the constraints expressing a geometric composition (based on the geometric union), with all the variants obtained by introducing selection conditions and spatial functions;
- the partition constraint between a class C_{whole} and a class C_{part} by means of a composition constraint $PW_{\uplus}(C_{\mathsf{whole}}, g, C_{\mathsf{part}}, f)$, plus a $TC^{b}_{\exists}(C_{\mathsf{part}}, f, \{\mathsf{IN} \mid \mathsf{EQ}\}, C_{\mathsf{whole}}, g)$ on the parts, for expressing their geometric containment in a whole, and a $TC^{b}_{\forall}(C_{\mathsf{part}}, f, \{\mathsf{DJ}\}, C_{\mathsf{part}}, f)$, for expressing the geometric disjunction among the parts.

Possible extensions of the set of constraints regards: (i) the introduction of selection condition depending on non local properties, (ii) the introduction of geometric functions on set of geometries, (iii) the introduction of geometric functions that compute distance-based properties. Every extension to the set of templates requires to specify the semantics of the template by an OCL expressions, the parameters of the template and the corresponding translation in the target technology (i.e. SQL in PostGIS, for example).

Future work will regard: (i) providing a detailed and formal demonstration about the possibility of implementing the 3D topological relations presented in Sect. 4.2 by combining a set of basic functions commonly available in GIS systems, such as PostGIS, together with a very restricted set of custom procedures; (ii) the extension of the templates to other cases, as specified above; (iii) the implementation of validator tools for city data stored in spatial DBMS; (iv) the testing of the proposed approach on huge datasets using a map-reduce framework, as done in [20] for 2D spatial constraint templates, (v) the extension of the proposed approach for the definition of integrity constraints on multi-accuracy spatial datasets [1].

References

1. Belussi, A., Migliorini, S.: A framework for integrating multi-accuracy spatial data in geographical applications. Geoinformatica **16**(3), 523–561 (2012)
2. Belussi, A., Migliorini, S., Negri, M., Pelagatti, G.: Validation of Spatial Integrity Constraints in City Models. In: Proc. of the 4th ACM SIGSPATIAL Int. Workshop on Mobile Geographic Information Systems, MobiGIS '15, pp. 70–79. ACM (2015)
3. Belussi, A., Negri, M., Pelagatti, G.: Modelling Spatial Whole-Part Relationships using an ISO-TC211 Conformant Approach
4. Belussi, A., Negri, M., Pelagatti, G.: 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, Proceedings of ER 2006, vol. 4231, pp. 100–109 (2006)
5. European Commission Joint Research Centre: INSPIRE Data Specification on Buildings - Technical Guidelines (2013). Version 3.0
6. Demuth, B., Hussmann, H., Loecher, S.: 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, pp. 104–117 (2001)
7. Diakité, A.A., Damiand, G., Gesquière, G.: Automatic semantic labelling of 3D buildings based on geometric and topological information. In: Proc. 9th International 3DGeoInfo Conference, pp. 49–63 (2014)
8. Duboisset, M., Pinet, F., Kang, M.A., Schneider, M.: Precise Modeling and Verification of Topological Integrity Constraints in Spatial Databases: From an Expressive Power study to Code Generation Principles. In: Conceptual Modeling - ER, pp. 465–482 (2005)
9. Egenhofer, M.J.: Topological relations in 3D. Tech. Rep. 05/1995, University of Maine, USA (1995)
10. Egenhofer, M.J., Franzosa, R.: Point-set topological spatial relations. International Journal of Geographic Information Systems **2**(5), 161–174 (1991)
11. Gröger, G., Plümer, L.: How to Achieve Consistency for 3D City Models. GeoInformatica **15**, 137–165 (2011)
12. Gröger, G., Plümer, L.: CityGML – Interoperable Semantic 3D City Models. ISPRS Journal of Photogrammetry and Remote Sensing **71**, 12–33 (2012)
13. Open Geospatial Consortium Inc.: OpenGIS Implementation Standard for Geographic Information - Simple feature access - Part 1: Common architecture (2011). Version 1.2.1 `http://www.opengeospatial.org/standards/sfa`

14. Open Geospatial Consortium Inc.: OGC City Geography Markup Language (CityGML) Encoding Standard (2012). Version 2.0 http://www.opengeospatial.org/standards/citygml
15. ISO: ISO 19107 Geographic Information – Spatial Schema (2003). https://www.iso.org/standard/26012.html
16. ISO: Geographic information – Geography Markup Language (GML) (2007). https://www.iso.org/standard/32554.html
17. ISO: ISO 19109 Geographic Information – Rules for application schema (2015). https://www.iso.org/standard/59193.html
18. Ledoux, H.: On the validation of solids represented with the international standards for geographic information. Comput. Aided Civ. Inf. **28**(9), 693–706 (2013)
19. Li, L., Luo, F., Zhu, H., Ying, S., Zhao, Z.: A two-level topological model for 3D features in CityGML. Computers, Environment and Urban Systems **59**, 11–24 (2016)
20. Migliorini, S., Belussi, A., Negri, M., Pelagatti, G.: Towards Massive Spatial Data Validation with SpatialHadoop. In: Proceedings of the 5th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, BigSpatial'16, pp. 18–27 (2016)
21. OGC: OGC CityGML quality interoperability experiment. Open Geospatial Consortium inc. (2016). URL https://portal.opengeospatial.org/files/?artifact_id=68821
22. Object Management Group (OMG): Object Constraint Language (OCL) (2014). Version 2.4 http://www.omg.org/spec/OCL/2.4/PDF/
23. Pelagatti, G., Negri, M., Belussi, A., Migliorini, S.: From the Conceptual Design of Spatial Constraints to Their Implementation in Real Systems. In: 17th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems, pp. 448–451 (2009)
24. Stadler, A., Kolbe, T.H.: Spatio-Semantic Coherence in the Integration of 3D City Models. In: 5th Int. ISPRS Symp. on Spatial Data Quality (ISSDQ), pp. 1–8 (2007)
25. Wagner, D., Wewetzer, M., Bogdahn, J., Alam, N., Pries, M., Coors, V.: Geometric-Semantical Consistency Validation of CityGML Models. In: Progress and New Trends in 3D Geoinformation Sciences, pp. 171–192 (2013)
26. Woo, T.C., Thomasma, T.: An algorithm for generating solid elements in objects with holes. Computers & Structures **18**(2), 333–342 (1984)
27. Xu, D.: 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)
28. Xu, D., van Oosterom, P., Zlatanova, S.: A methodology for modelling of 3D spatial constraints. In: Advances in 3D Geoinformation, pp. 95–117 (2016)