

**AEC 3**

# **IFG Project**

**Phase 1**

## **Comparison of gml3.0 and IFC2x(2)**

Thomas Liebich

AEC3 Ltd.

Munich, Thatcham

27 May 2004

## ***Acknowledgement***

This document has been developed for the IFG (Industry Foundation Classes for GIS) Project, funded by the National Office of Building Technology and Administration, Oslo, Norway.

## ***Table of Content***

<b>1</b>	<b>COMPARISON OF FEATURE MODEL SCHEMA</b>	<b>5</b>
<b>2</b>	<b>COMPARISON OF GEOMETRY SCHEMA</b>	<b>8</b>
2.1	Coordinate geometry	9
2.2	Geometric Complex and geometric composites	10
2.3	Geometric Primitives (0- and 1-dimensional)	10
2.4	Geometric Primitives (2-dimensional)	18
2.5	Geometric Primitives (3-dimensional)	22
2.6	Geometric aggregates	22
2.7	Geometric properties	24
2.8	User-defined Geometry Types and Geometry Property Types	25
<b>3</b>	<b>COORDINATE REFERENCE SYSTEMS</b>	<b>26</b>
<b>4</b>	<b>COMPARISON OF TOPOLOGY SCHEMA</b>	<b>32</b>

## ***Table of Figures and Tables***

FIGURE 1: FEATURE MODEL IN GML3.0	6
FIGURE 2: GEOMETRY OVERVIEW IN GML3.0	9
FIGURE 3: HIERARCHY OF GEOMETRY ELEMENTS IN GML3.0	11
FIGURE 4: GML3.0 DEFINITION OF CURVE AND CURVE SEGMENTS	14
FIGURE 5: MORE GML3.0 DEFINITION OF CURVE AND CURVE SEGMENTS	16
FIGURE 6: GML3.0 DEFINITION OF ORIENTABLE CURVE AND COMPOSITE CURVE	17
FIGURE 7: GML3.0 DEFINITION OF POLYGON	19
FIGURE 8: GML3.0 DEFINITION OF SURFACE AND SURFACE PATCHES	19
FIGURE 9: GML3.0 DEFINITION FOR ORIENTABLE AND COMPOSITE SURFACE	21
FIGURE 10: GML3.0 DEFINITION OF SOLID	22
FIGURE 11: GML3.0 DEFINITIONS OF GEOMETRIC AGGREGATES	23
TABLE 1: LIST OF SHAPE REPRESENTATION TYPES IN IFC	24
TABLE 2: GEOMETRIC PROPERTIES IN GML3.0	25

# 1 Comparison of feature model schema

The equivalent of an *gml:\_Feature* is an *lfcProduct* within the IFC2x(2) schema. The following table compares both definitions.

In gml3.0 a feature is:

**“a meaningful object in the selected domain of discourse such as a Road, River, Person, Vehicle or Administrative Boundary.”**

A *\_Feature* is characterised by:

- (multiple) name(s), description, id and metadata properties (inherited from *\_GML*)
- boundary (by various means), describes an envelope that encloses the entire feature instance
- location (by various means), describes the extent, position or relative location of the feature.

A *\_Feature* (and the corresponding type *AbstratFeatureType*) is meant as an abstract element and type for application schema writers to generate their specific features

A *boundedBy* (and *BoundingShapeType*) defines the envelop that encloses the entire feature, is can either be a *Null* (not presented) or an *Envelop* (with the special substitution of an Envelop with time period).

A *location* (and *LocationPropertyType*) defines various possibilities to position a feature. It includes geometry, location string and key word).

There exists the possibility to assign a priority to the (potentially multiple) locations by use of *PriorityLocationPropertyType*

in IFC2x(2) a product is:

**Any object, or any aid to define, organize and annotate an object, that relates to a geometric or spatial context.**

An *lfcProduct* is characterised by:

- id, owner history, name, description (inherited from *lfcRoot*), and property data links and associations (from *lfcObject*)
- object placement (by relative or absolute Cartesian axis placement)
- representation, single or multiple representations, including geometric representations of the product

An *lfcProduct* is the supertype for all shared or domain specific entities describing products in particular areas, like wall, valve, furniture, etc.)

One of the representations within the representation attribute link, (*lfcShapeRepresentation*) is the *BoundingBox* representation, which provides the envelop for the 3D shape (a bounding rectangle can be added to the spec for 2D shape) . A time dependent envelop description does not exist in IFC.

The *ObjectPlacement* defines the position of a feature in terms of geometry, either as a global placement (within WCS), or local placement (relative to another object placement) or as a grid placement. Location by string or key word does not exist in IFC.

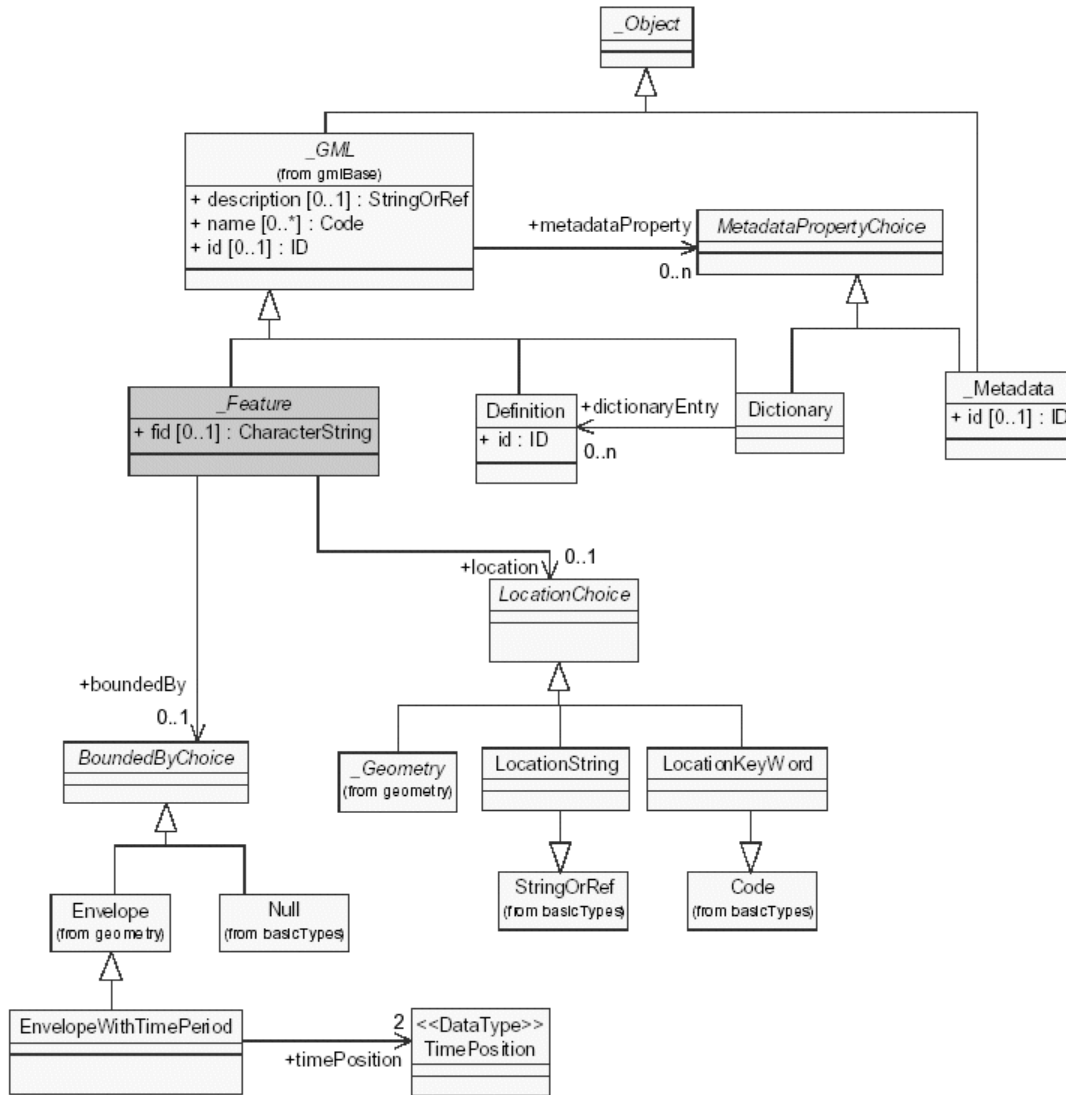


Figure 1: feature model in gml3.0

Features can have associations to feature members and feature properties. This is handled by the complexType *FeaturePropertyType* and the two elements *featureMember* and *featureProperty*. Similar associations exist in the IFC2x(2) model, *lfcRelAssigns*, *lfcRelDefines*, *lfcRelDecomposes*, *lfcRelAssociates*. In contrary to gml3.0 the IFC association types describe two objectified relationships, whereas the gml associations defined abstract elements, that provide the link to the feature and can be substituted to contain the actual content model for the member or property part.

A *featureProperty* is an abstract element (type *FeaturePropertyType*) that has the link to an *\_Feature*. Similarly a *featureMembers* (type *FeatureArrayPropertyType*) handles the association of a property to many features.

An *lfcRelDefinesByProperties* is used to associate a single property set (predefined or generic) to a product or multiple products. It therefore covers both (a single occurrence association and a multiple occurrence association)

A *featureMember* is an abstract element (type *FeaturePropertyType*) that is used to build a collection of features (to act as a single feature). The resulting feature collection is described as *FeatureCollection* (with type *AbstractFeatureCollectionType* and derived *FeatureCollectionType*). It is a substitution of a bounded feature (*BoundedFeatureType*) that has an association to *featureMember(s)*.

A *FeatureCollection* is itself a valid feature. It therefore relates to *IfcRelDecomposes* in IFC2x(2) (and not to the IFC grouping mechanism).

There is a difference in gml3.0 of *featureMember* within a *FeatureCollection* allowing for reference of “remote” features, whereas a *featureMembers* require an “inline” definition of the members.

The IFC2x(2) describe two related constructs.

Any *IfcObject* itself can be defined by other objects and thereby be decomposed. There are two kinds of decomposition: an aggregation and a nesting. Aggregation (by *IfcRelAggregates*) allows for an object to be decomposed in any set of semantically different objects, whereas nesting (by *IfcRelNests*) requires the members to be of the same type as the collection.

An *IfcGroup* is a collection of any object (including products and other groups) by virtue of the *IfcRelAssignsToGroup* relationship that can be logically grouped. However the group itself does not have a particular meaning (other than being a collection).

IFC2x(2) and ifcXML does not make a distinction between reference and containment.

## 2 Comparison of geometry schema

The geometry in gml3.0 is currently structured according to the following schema breakdown:

- geometryBasic0d1d.xsd
- geometryBasic2d.xsd
- geometryPrimitives.xsd
- geometryAggregates.xsd
- geometryComplexes.xsd

All geometry classes inherit an optional reference to a coordinate reference system. All direct positions shall directly or indirectly be associated with a coordinate reference system. When geometry elements are aggregated in another geometry element (such as a MultiGeometry or GeometricComplex), which already has a coordinate reference system specified, then these elements are assumed to be in that same coordinate reference system unless otherwise specified.

The IFC geometry resources have been taken from the Integrated Resource, part 42 "Integrated generic resources: Geometric and topological representations" of the ISO standard 10303: "Industrial automation systems and integration - Product data representation and exchange".

The definitions taken from ISO/IS 10303-42:1994 have undergone a adaptation process, characterised by:

- adaptation of the IFC naming convention (inner majuscules and lfc prefix)
- adaptation of the STEP entities, where multiple inheritance or non-exclusive inheritance (i.e. AND or ANDOR subtype constraints) are used
- selection of a subset of the IR, using subtype and select pruning
- dimensionality of geometric representation items defined at each item (not through the representation context)
- omission of pcurves, use of simple 2D curves for the generation of swept surfaces
- omission of the name attribute at the representation item

In IFC2x(2) a similar construct is given by the *lfcGeometricRepresentationContext*. However the reference is not direct from the geometric item, but through the *lfcShapeRepresentation*, the container for all geometric representation items for a given view of a product. Due to the ability of relative local placements it is assumed, that all representations, that are given relative to the object placement of other products, are in that same coordinate reference system.

*Note: Usually the **lfcSite** is positioned directly in that coordinate reference system, and all other products are placed relative (often cascading) to the object placement of the **lfcSite** object.*

The geometry model distinguishes geometric primitives, aggregates and complexes plus composite geometries.

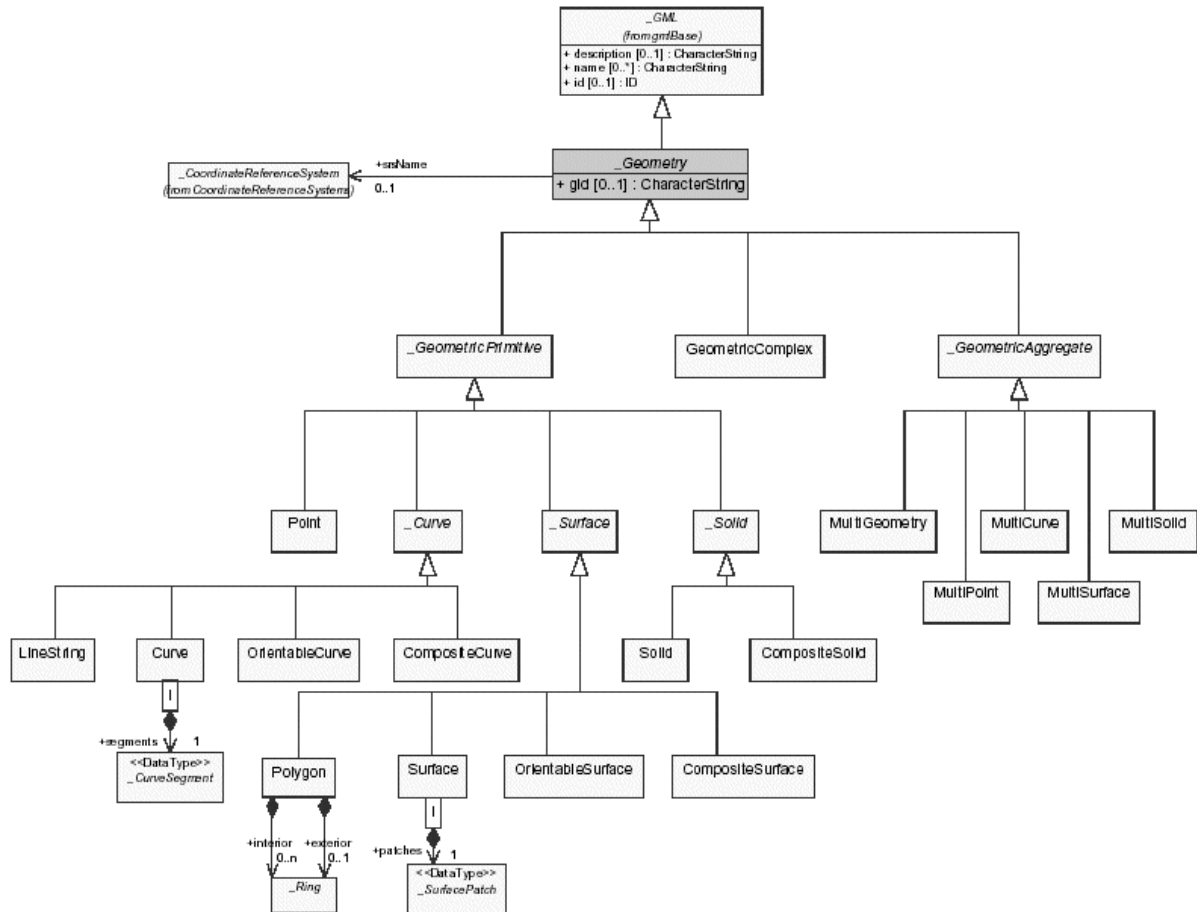


Figure 2: Geometry overview in gml3.0

An *\_Geometry* element (and complexType *AbstractGeometryType*) is the abstract head of all geometry elements. It inherits from *\_GML*:

- (multiple) name(s), description, id and metadata properties

It has an optional link to the coordinate reference system (attribute *srsName*).

A *GeometryPropertyType* is a type encapsulating a geometry element, a *GeometryArrayPropertyType* encapsulates several geometry elements.

The *IfcGeometricRepresentationItem* is the base class for all geometry classes. An geometric representation item is a representation item that has the additional meaning of having geometric position or orientation or both. It does not have any attributes.

Therefore the IFC geometric items do not have the ability to carry the overhead of name, description, id or metadata.

The link to the coordinate reference system is handled differently (see text above).

There is no direct equivalence of such encapsulating type, beside the *IfcRepresentation* and *IfcShapeRepresentation* which are used to include on or more geometric representation items for the representation of an object's shape.

## 2.1 Coordinate geometry

The coordination geometry is defined in the geometryBasic0d1d.xsd, it contains the basic geometry items that provide position, direction, etc.

A *DirectPositionType* holds the coordinates for a position within some coordinate reference system. It is independent of its dimensionality.

The definition of coordinates in IFC2x(2) is given by the *IfcCartesianPoint*. It is also independent of its dimensionality. In contrary to gml3.0 it is independent of its usage as well (e.g. can be used for position, start- end point, etc.)

In contrary to gml3.0 *IfcCartesianPoint* is independent of its usage as well (e.g. can be used for position, start- end point, etc.). It covers both.

An element *coordinates* also provides for the provision of coordinates (with free delimiter, whereas *DirectPositionType* is of type xs:list and has whitespace as delimiter.

IFC2x(2) does not make differences between different encodings. In ifcXML all lists are handled as sequences of elements (sparse representation).

An element *pointRep* allows both encodings as representations of a point.

Equal to *IfcCartesianPoint* (but making no difference regarding the encoding)

A *vector* is an ordered set of numbers called coordinates that represent a position in a coordinate reference system.

In IFC2x(2) an *IfcVector* has an additional magnitude and the orientation values are only given as ratios (but measure values)

An *envelope* is often referred to as a minimum bounding box or rectangle. Regardless of dimension, an envelope can be represented without ambiguity as two direct positions (coordinate points).

In IFC2x(2) the same is provided for 3D envelopes as *IfcBoundingBox*. It is given differently, by a position and 3 extents. For 2D envelopes it is provided by *IfcPlanarBox* (again by position and 2 extents).

## 2.2 Geometric Complex and geometric composites

A geometric complex is a set of primitive geometric objects (in a common coordinate system) whose interiors are disjoint.

A geometric composite (GML specifies *CompositeCurve*, *CompositeSurface* or *CompositeSolid*) represents a geometric complex with an underlying core geometry that is isomorphic to a primitive. Thus, a composite curve is a collection of curves whose geometry could be viewed as a curve (albeit one with a complex inner structure). Composites are intended for use as attribute values in data sets in which the underlying geometry has been decomposed, usually to expose its topological nature.

Geometric composite are defined as substitutions of the relevant geometric primitives (*\_Curve*, *\_Surface*, *\_Solid*) and are described there. In general there is no direct equivalence in IFC2x(2) for such geometric composites as geometric items. But IFC2x(2) offers topological items that have the same capabilities – so the use of *IfcPath*, *IfcConnectedFaceSet* would be appropriate for composite curves and surfaces. For composite solids the equivalence is *IfcShellBasedSurfaceModel*.

The base definitions, *GeometricComplexType* and *GeometricComplexPropertyType* are used to reference geometric complex and geometric composites. They do not have an equivalence in IFC2x(2).

## 2.3 Geometric Primitives (0- and 1-dimensional)

Geometry elements are first structures according to the criteria: primitive, complex, aggregate. Then the primitives are structured according to the base types, point, curve, surface, solid. In IFC2x(2) the structuring of subtypes is opposite, first the geometric types, and then for each type, whether it is primitive, complex or an aggregate.

A *\_GeometricPrimitive* (type *AbstractGeometricPrimitiveType*) is the abstract head of the substitution group for all (pre- and user-defined) geometric primitives.

An *GeometricPrimitivePropertyType* defines a type that encapsulates the reference or containment of a *\_GeometricPrimitive*.

There is no equivalent in IFC2x(2), as the subtyping structure is different. However since the gml *\_GeometricPrimitive* does not define an extended content model, it is not critical.

There is no direct equivalence of such encapsulating type, beside the *lfcRepresentation* and *lfcShapeRepresentation* which are used to include on or more geometric representation items for the representation of an object's shape.

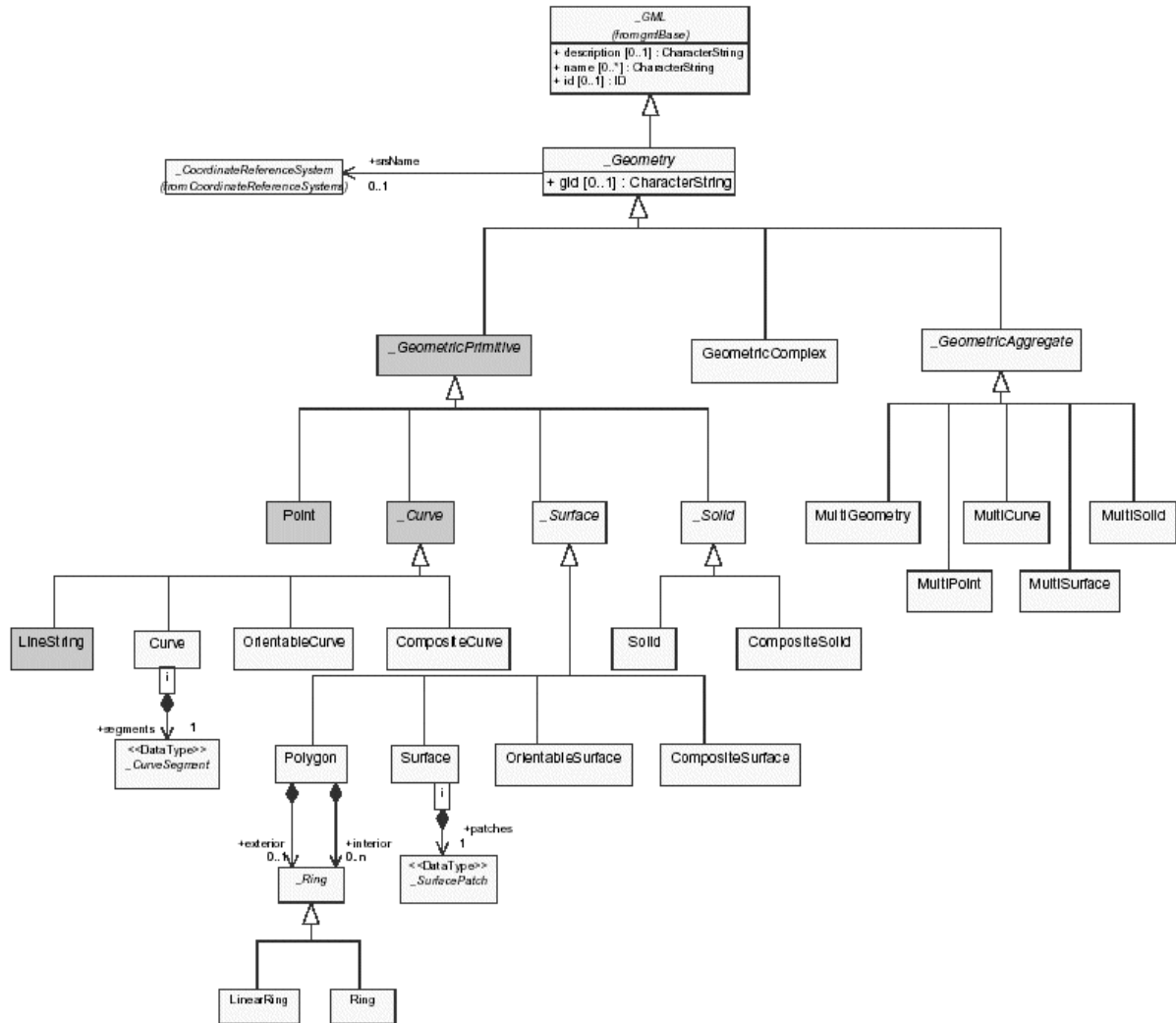


Figure 3: Hierarchy of geometry elements in gml3.0

There are four basic geometric primitives in gml3.0, point, curve, surface, solid. This is equivalent to the *lfcPoint*, *lfcCurve*, *lfcSurface* and the more detailed solid part with *lfcSolidModel*, *lfcShellBasedSurfaceModel*, *lfcFaceBasedSurfaceModel* and *lfcBooleanResult*.

The Figure 3 shows the breakdown in gml3.0. The 0-dimensional elements in gml3.0 is the point.

A *Point* (type *PointType*) is a point in some coordinate space. It can be given by *pos* or *coordinate* element.

The equivalent in IFC2x(2) is the abstract *lfcPoint* and the *lfcCartesianPoint*.

Note: The different to *pointRep* is unclear, as both definitions seem to be the same.

Points can be used as geometry properties in *PointProperty* (type *PointPropertyType*) and *PointArrayProperty* (type *PointArrayPropertyType*)

There is no direct equivalence of such encapsulating type, see also the general note.

1-dimensional elements in gml3.0 comprises (poly)line (as line string), curve, orientable curve, and composite curve. The most simple type is line string.

A *\_Curve* (type *AbstractCurveType*) can always be viewed as a geometric primitive, i.e. is continuous.

The equivalent in IFC2x(2) is the abstract *IfcCurve* that has many subtypes.

Curves can be used as geometry properties in *curveProperty* (type *CurvePropertyType*) and *curveArrayProperty* (type *CurveArrayPropertyType*)

There is no direct equivalence of such encapsulating type, see also the general note.

A *LineString* (type *LineStringType*) is a special curve that consists of a single segment with linear interpolation. It is defined by two or more coordinate tuples, with linear interpolation between them. It can be given by *pos*, *pointRep* or *coordinate* elements.

The equivalent in IFC2x(2) is the *IfcPolyLine*. It is represented by a list of 2 or more Cartesian points.

Beside the simple line string, there are more complex definitions of curves.

A *Curve* (type *CurveType*) is a 1-dimensional primitive. Curves are continuous, connected, and have a measurable length in terms of the coordinate system.

The equivalent in IFC2x(2) is *IfcCompositeCurve*, which is a collection of curves joined end-to-end. The individual segments of the curve are themselves defined as composite curve segments.

A curve is composed of one or more curve segments. Each curve segment within a curve may be defined using a different interpolation method. The curve segments are connected to one another, with the end point of each segment except the last being the start point of the next segment in the segment list. The orientation of the curve is positive.

Note: in gml3.0 there is a *CompositeCurve* as well, which is a sequence of Curve's.

The segments of the *IfcCompositeCurve* are given by *IfcCompositeCurveSegment*.

The element "segments" encapsulates the segments of the curve.

The abstract *\_CurveSegment* (type *AbstractCurveSegmentType*) defines a homogeneous segment of a curve.

The equivalent in IFC2x(2) is *IfcCompositeCurveSegment*, however it is not an abstract type and not subtyped according to the different geometry kinds of curve segments. An *IfcCompositeCurveSegment* has a reference to the underlying curve (*ParentCurve :: IfcCurve*) that handles the different ways a homogeneous segment can be defined.

In addition there is a container for curve segment types, the *CurveSegmentArrayPropertyType*.

It has three attributes dealing with the continuity of the segments, *numDerivativesAtStart*, *numDerivativesAtEnd*, *numDerivativesInterior*.

The element *Segment* uses the *CurveSegmentArrayPropertyType* to define an element that contains a list of curve segments. The order of the elements is significant.

All substitution elements of *\_CurveSegment* have an attribute *CurveInterpolationType* that determines the interpolation mechanisms specified by an application schema.

The following interpolations are available:

- linear
- geodesic
- circularArc3Points
- circularArc2PointWithBulge
- circularArcCenterPointWithRadius
- elliptical
- clothoid
- conic
- polynomialSpline
- cubicSpline
- rationalSpline

It has an attribute *Transition* that deals with the continuity of segments, always describing the geometric continuity from the last point of this segment to the first point of the next segment (it relates to the *numDerivativesAtStart*, *numDerivativesAtEnd*, without the redundancy as in gml3.0).

The equivalent in IFC2x(2) is *IfcCompositeCurve* that has a list (order significant) of *IfcCompositeCurveSegment*.

In IFC2x(2) the curve interpolation is given by the type of the *IfcCurve* referenced as *ParentCurve*.

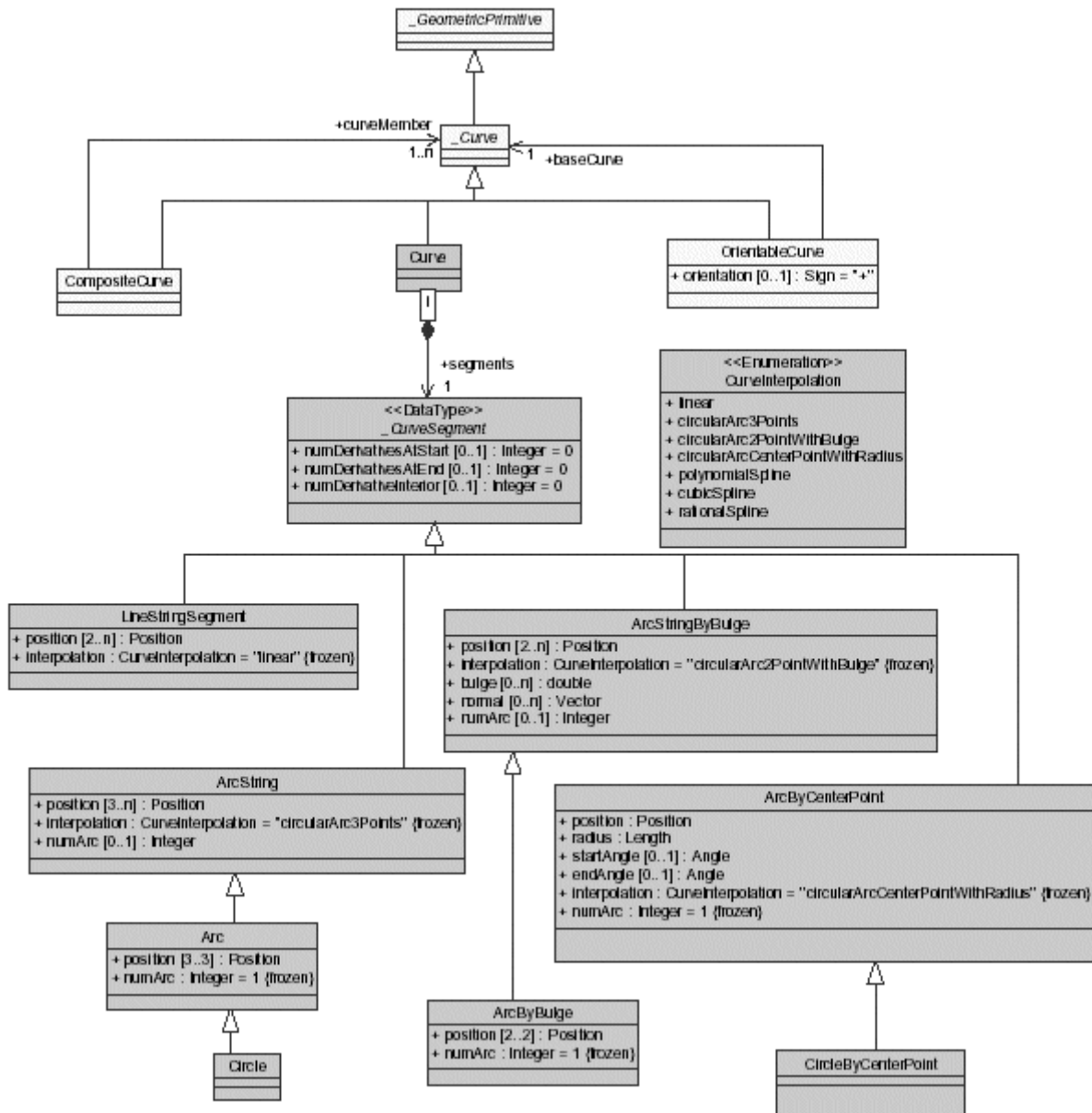


Figure 4: gml3.0 definition of curve and curve segments

The following segment types define the substitution groups of the abstract *\_CurveSegment*.

*Note: In contrary to gml3.0 each of the curve segment types in IFC are independent classes that can be instantiated without being part of a curve with segments, e.g. an arc could be directly expressed by an *IfcTrimmedCurve* without being included into an *IfcCompositeCurve*.*

An *LineStringSegment* (type *LineStringSegmentType*) is a curve segment that is defined by two or more coordinate tuples, with linear interpolation between them. It can be defined using the same alternative mechanisms as the *LineString* (see above).

The equivalence in IFC2x(2) is an *IfcCompositeCurveSegment* referencing an *IfcPolyline* by *ParentCurve*.

An *ArcString* (type *ArcStringType*) is a series of curve segment that uses three-point circular arc interpolation. A point can be specified either by *pos*, *pointRep* or *coordinates* elements.

An *Arc* (type *ArcType*) is a curve segment that uses three-point circular arc interpolation. A point can be specified either by *pos*, *pointRep* or *coordinates* elements.

A *Circle* (type *CircleType*) is an arc whose first and last control points coincide to form a full circle. It is a substitution of *Arc*.

An *ArcStringByBulge* (type *ArcStringByBulgeType*) is variant of the arc that computes the mid points of the arcs instead of storing the coordinates directly. It can have multiple segments.

An *ArcByBulge* (type *ArcByBulgeType*) is variant of the arc that computes the mid point of the arcs instead of storing the coordinates directly. It can have only a single segment.

An *ArcByCenterPoint* (type *ArcByCenterPointType*) is a variant of the arc that requires that the points on the arc have to be computed instead of storing the coordinates directly. The control point is the center point of the arc plus the radius and the bearing at start and end.

The element “*radius*” specifies the radius of the arc. The element “*startAngle*” specifies the bearing of the arc at the start. The element “*endAngle*” specifies the bearing of the arc at the end.

This representation can be used only in 2D.

A *CircleByCenterPoint* (type *CircleByCenterPointType*) is a circle whose start and end angle coincide to form a full circle. It is a substitution of *ArcByCenterPoint*.

There is no direct equivalence in IFC2x(2), it needs to be broken up into a series of *IfcCompositeCurveSegment*. Each segment is then defined as shown below.

There is no direct equivalence in IFC2x(2). It can be defined by an *IfcCompositeCurveSegment* referencing an *IfcTrimmedCurve* by *ParentCurve*. The *IfcTrimmedCurve* has to reference an *IfcCircle* by *BasisCurve*.

There are two ways to define the start and end point of a trimming: by parameter or by Cartesian point. The equivalence to gml3.0 would be the use of *IfcCartesianPoint* for *Trim1* and *Trim2*.

The Center point and radius have to be given as well, therefore a 3 point arc needs to be converted first into an arc by center point.

In IFC2x(2) an *IfcTrimmedCurve* (with identical start & end points has to be used, like in the arc example before. That is identical with the usage in gml3.0

But note, that an independent circle is defined in IFC2x(2) as *IfcCircle* with no subtyping relationship to *IfcTrimmedCurve*.

In IFC2x(2) there is no equivalent to this type. It needs to be converted into a series of arcs given by three explicit control points.

In IFC2x(2) there is no equivalent to this type. It needs to be converted into an arc given by three explicit control points.

The equivalence in IFC2x(2) is more complex. It is defined by an *IfcCompositeCurveSegment* referencing an *IfcTrimmedCurve* by *ParentCurve*. The *IfcTrimmedCurve* has to reference an *IfcCircle* by *BasisCurve*.

There are two ways to define the start and end point of a trimming: by parameter or by Cartesian point. The equivalence to gml3.0 would be the use of *IfcParameterValue* for *Trim1* and *Trim2*. The *Radius* attribute specifies the radius.

In contrary to gml3.0 the IFC definition can be used in 2D and 3D as it is fully determined by the parameterization of the underlying circle.

In IFC2x(2) an *IfcTrimmedCurve* (with identical start & end parameter values has to be used, like in the arc by center point example before. That is identical with the usage in gml3.0

But note, that an independent circle is defined in IFC2x(2) as *IfcCircle* with no subtyping relationship to *IfcTrimmedCurve*.

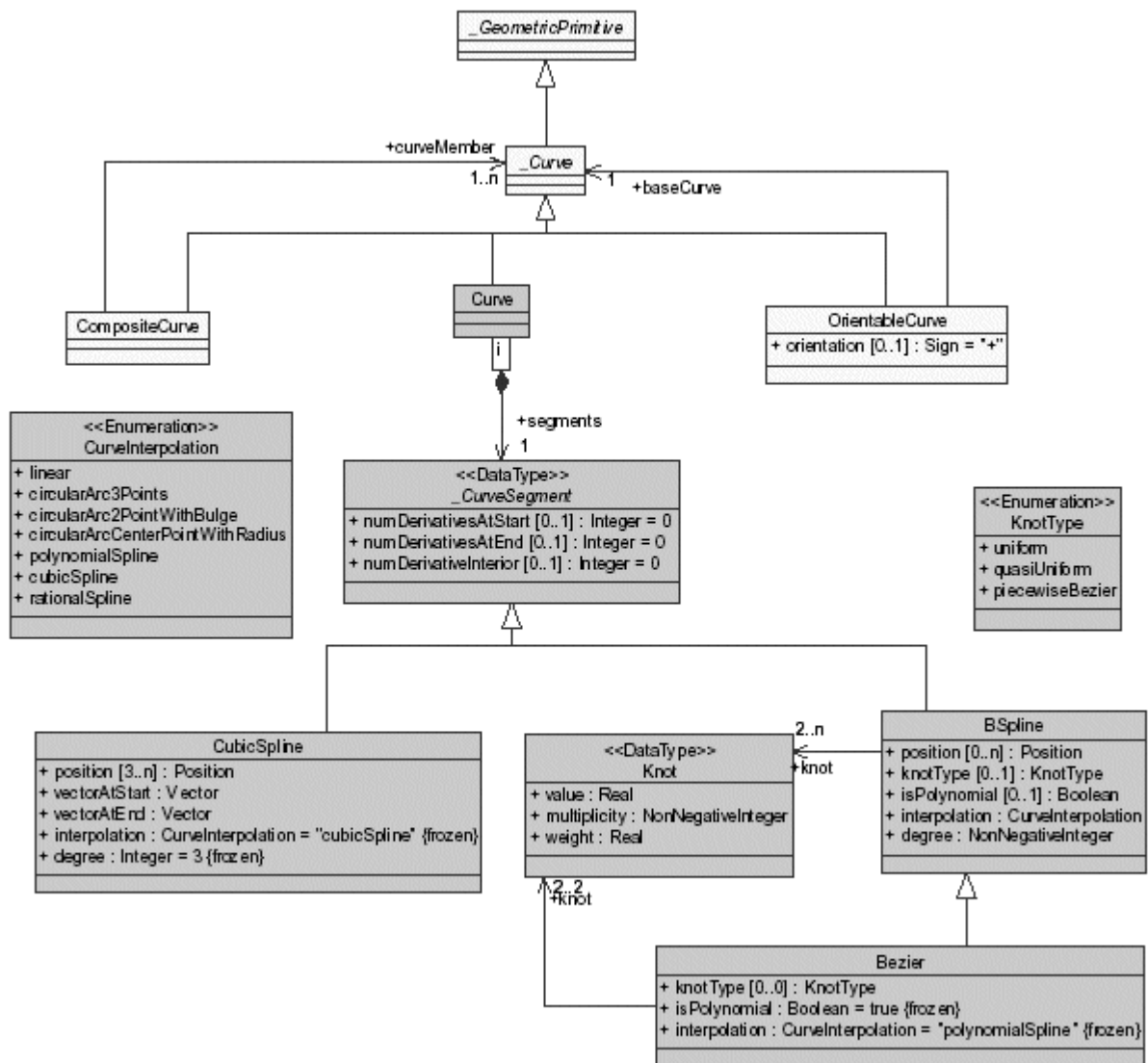


Figure 5: more gml3.0 definition of curve and curve segments

Additionally the gml3.0 describes spline curves. Recently two basic forms of splines have been added to IFC2x2, the Bezier curve and the rational Bezier curve, however these are (in contrary to most other geometry objects) not part of the implemented coordination view subset.

*Note: There might be more effort to be spent to make a comprehensive comparison between gml3.0 and IFC and (as a source of IFC geometry definitions) STEP Part 42 spline definitions*

A **CubicSpline** (type **CubicSplineType**) is similar to line strings in that they are a sequence of segments each with its own defining function. A cubic spline uses the control points and a set of derivative parameters to define a piecewise 3rd degree polynomial interpolation. Unlike line-strings, the parameterisation by arc length is not necessarily still a polynomial.

It has a control vector at start and end.

In IFC2x(2) only two forms of splines are used:

- [IfcBezierCurve](#)
- [IfcRationalBezierCurve](#)

There is no direct equivalence to the **CubicSpline**.

A *BSpline* (type *BSplineType*) is a piecewise parametric polynomial or rational curve described in terms of control points and basis functions. Knots are breakpoints on the curve that connect its pieces.

The knots as breakpoints are mandatory definitions in gml3.0 given by the *KnotType*.

A *Bezier* (type *BezierType*) is a polynomial spline that use Bezier or Bernstein polynomials for interpolation purposes. It is a special case of the B-Spline curve with two knots.

The IFC2x2 *IfcBSplineCurve* is an abstract type and the only instantiable subtype is a Bezier curve. There is no type with mandatory knots in IFC.

Note: The Part 42 entity *b\_spline\_curve\_with\_knots* is not part of IFC yet.

The equivalence in IFC2x2 is *IfcBezierCurve*. Here suitable default values for the knots and knot multiplicities are derived in this case.

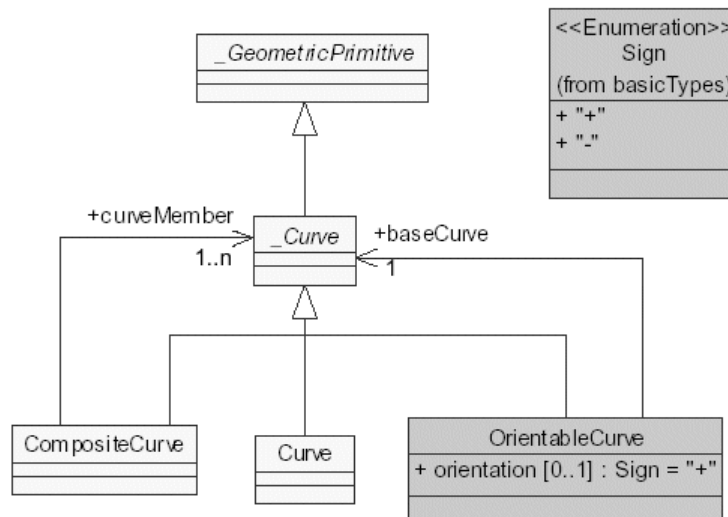


Figure 6: gml3.0 definition of orientable curve and composite curve

In addition to the Curve (as a composite of curve segments) there are two more general curve types: orientable curves and composite curves.

*OrientableCurve* (type *OrientableCurveType*) consists of a curve and an orientation. If the orientation is "+", then the *OrientableCurve* is identical to the *baseCurve*. If the orientation is "-", then the *OrientableCurve* is related to another *\_Curve* with a parameterisation that reverses the sense of the curve traversal.

A *CompositeCurve* (type *CompositeCurveType*) is defined by a sequence of (orientable) curves such that the each curve in the sequence terminates at the start point of the subsequent curve in the list.

In IFC2x(2) the handling of orientation is done on the level of subtypes of curve, not as a general class that adds orientation to any curve.

For trimmed curves it is handled by the *SenseAgreement* flag of *IfcTrimmedCurve*, for composite curves, each segment, defined as *IfcCompositeCurveSegment* has a flag *SameSense*. Both can revert the orientation of the base curve.

For all other curve subtypes there is no general way to revert the orientation.

There is no direct equivalence in IFC2x(2), as the *IfcCompositeCurve* is a collection of curve segments and therefore equivalent to *Curve*.

However the *IfcPath* (as a topology item) would offer the same capabilities when using it with a geometric definition of the edges by *IfcEdgeCurve*.

## 2.4 Geometric Primitives (2-dimensional)

The 2-dimensional geometric primitives comprise polygons and surface types, like surface, orientable surface and composite surface.

A *\_Surface* (type *AbstractSurfaceType*) is an abstraction of a surface to support the different levels of complexity. A surface is always a continuous region of a plane.

Surfaces can be used as geometry properties in *surfaceProperty* (type *SurfacePropertyType*) and *surfaceArrayProperty* (type *SurfaceArrayPropertyType*)

A *Polygon* (type *PolygonType*) is a special surface that is defined by a single surface patch. The boundary of this patch is coplanar and the polygon uses planar interpolation in its interior.

It has one outer (*exterior*) and zero to many inner (*interior*) boundaries. The elements *exterior* and *interior* provide for that. They are of type *AbstractRingPropertyType*, i.e. they handle the link to *\_Ring* as a boundary.

The polygon outer and inner boundaries are described by a *\_Ring* (type *AbstractRingType*). The provides *AbstractRingPropertyType* for the usage of *\_Ring* in other elements,

A *LinearRing* (type *LinearRingType*) is defined by four or more coordinate tuples, with linear interpolation between them; the first and last coordinates must be coincident.

The tuples can be given by *pos*, *pointRep* or *coordinate* elements.

The *LinearRingPropertyType* is used to encapsulates a ring to represent properties in features or geometry collections.

See also the following for the definition of polygon.

The equivalent in IFC2x(2) is the abstract *IfcSurface* that has many subtypes.

There is no direct equivalence of such encapsulating type, see also the general note.

The equivalent in IFC2x(2) is *IfcFaceSurface* (and not *IfcPolyLoop* or *IfcPolyline*, as the name might imply). The definition of *IfcFaceSurface* is more complex, it references an *IfcSurface* for the definition of the bounded area (or patch). Only if the type of the referenced *IfcSurface* is *IfcPlane*, it matches with the gml3.0 polygon. As *IfcFaceSurface* has a sense attribute it is orientable.

It references a set of bound (*IfcFaceBound*), one of which needs to be of type *IfcFaceOuterBound*. This relates to the gml3.0 *exterior* and *interior*.

There is no abstract ring (or face bound) type in IFC2x(2), but the *IfcFaceBound* and *IfcFaceOuterBound* provide for the functionality of a linear ring.

The equivalent in IFC2x(2) is *IfcPolyLoop* (as a special subtype of *IfcLoop*). There are other subtype, that have no equivalence in gml3.0 (*IfcVertexLoop*, *IfcEdgeLoop*).

The coordinates are always given by Cartesian points.

Note: In IFC2x(2) there is no need to specify the last coordinate being the same as the first. The loop is always be considered and being closed. Therefore the closing segment is from the last to the first point in the collection.

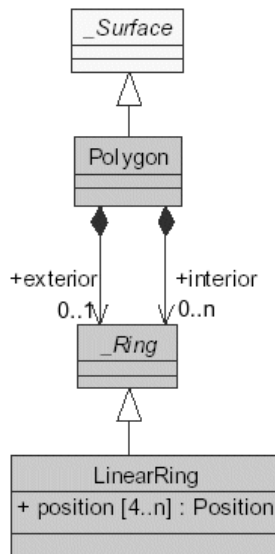


Figure 7: gml3.0 definition of polygon

There are other more complicated definitions of surfaces in gml3.0. A general surface element allows for connected surface sets, or a connection of surface patches.

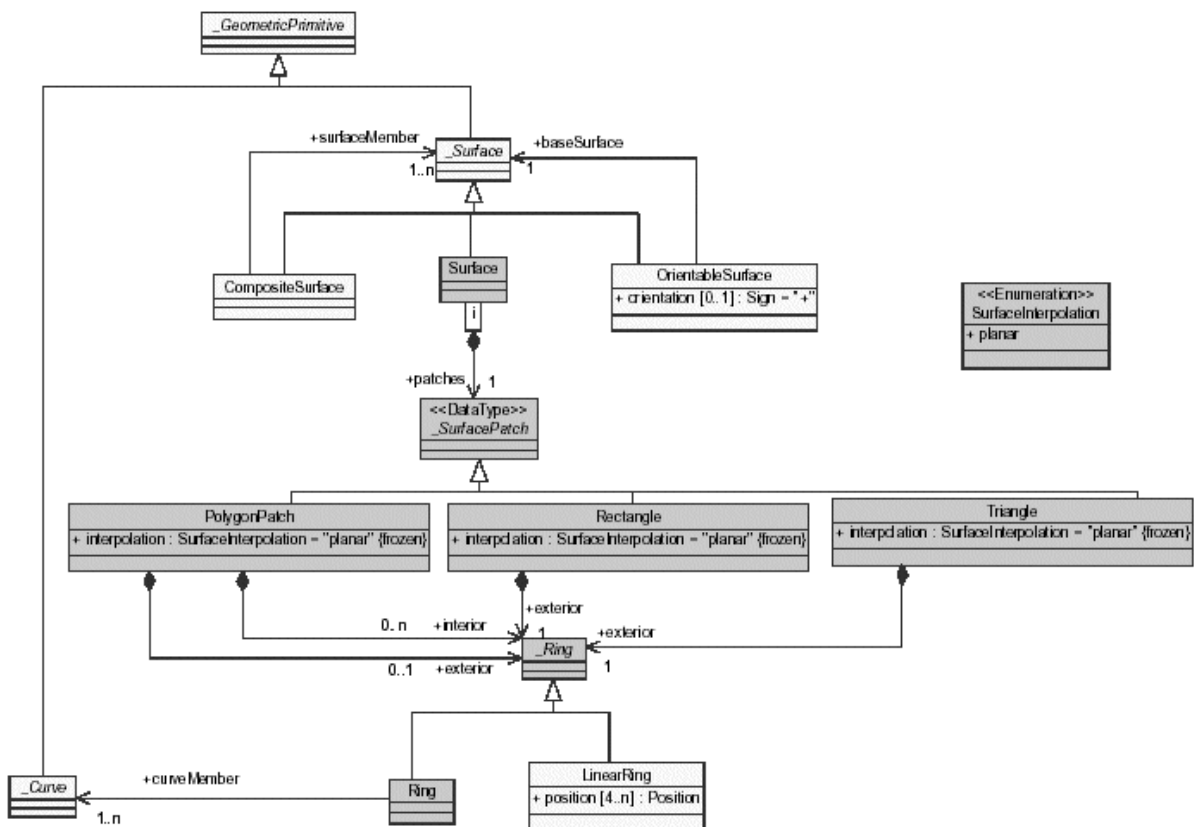


Figure 8: gml3.0 definition of surface and surface patches

A *Surface* (type *SurfaceType*) is a 2-dimensional primitive and is composed of one or more surface patches. The surface patches are connected to one another.

There is no direct equivalence in IFC2x(2) that describes a connected set of surfaces as geometric representation items.

A *\_SurfacePatch* (type *AbstractSurfacePatchType*) is the supertype of all surface patches in gml3.0. The element *patches* (type *SurfacePatchArrayPropertyType*) defines a sequence of *\_SurfacePatch* that describes the patches within a *Surface* element.

The *SurfaceInterpolationType* defines the different interpolations used for surface definitions in gml3.0. It includes:

- none
- planar
- spherical
- elliptical
- conic
- tin
- parametricCurve
- polynomialSpline
- rationalSpline
- triangulatedSpline

There are three non-abstract subtypes of *\_SurfacePatch*. They include polygonal patches, rectangular patches and triangular patches. In IFC2x(2) there are no entities describing a surface patch. It could be handled by two ways:

1. a set of surfaces within an *IfcGeometricSet*.
2. another possibility is to use a shell based representation, using an *IfcShellBasedSurfaceModel* with an *IfcOpenShell*.

A *PolygonPatch* (type *PolygonPatchType*) is a surface patch that is defined by a set of boundary curves and an underlying surface to which these curves adhere. The curves are coplanar and the polygon uses planar interpolation in its interior.

It has one outer boundary and may have several inner boundaries.

A *Triangle* (type *TriangleType*) is a surface patch in form of a triangle. It represents a triangle as a surface with an outer boundary consisting of a linear ring having four points, the first and the last must be co-incident. It does not have an inner boundary.

An *Rectangle* (type *RectangleType*) is a surface patch in form of rectangle. It represents a rectangle as a surface with an outer boundary consisting of a linear ring having five points, the first and the last must be co-incident. It does not have an inner boundary.

The closest equivalence is the topological representation item *IfcConnectedFaceSet*. It is a set of faces such that the domain of faces together with their bounding edges and vertices is connected.

There is no equivalence within IFC2x(2) to express surface patches in a geometric way. Currently the topological item *IfcConnectedFaceSet* is used.

NOTE: Part42 3<sup>rd</sup> edition now defines *surface\_patch* – it could be used to extend IFC.

Currently there are no surface patches in IFC. However IFC defines various surface definitions that could be used as individual surface definitions. It includes planar surface.

As said above, IFC2x(2) does not have built in support for surface patches. However the entity *IfcCurveBoundedPlane* offers an identical definition for polygonal surfaces.

A set of such could be used within an *IfcGeometricSet* to describe the same form, but without the connectivity implied by a surface patch.

There is no such special surface entity in IFC2x(2), it could be represented by *IfcCurveBoundedPlane* using the constraints enforced by a *Triangle*.

There is an entity in IFC2x(2) to represent rectangular, planar surfaces, the *IfcRectangularTrimmedSurface*, using a *BasisSurface* of type *IfcPlane*.

However the definition is different and uses the parameterisation of the plane. It has to be orthogonal, which does not seem to be the case for the *Rectangle*.

Therefore it should be represented by *IfcCurveBoundedPlane* using the constraints enforced by a *Rectangle*.

In addition gml3.0 has several property element definitions, that are needed for the gml3.0 XSD notation, but don't need to have an equivalence in IFC2x(2). These are *curveMember*, *Ring* (type *RingType*), *baseSurface*.

In addition to the *\_Surface* there are two more general surface elements: *OrientableSurface* and *CompositeSurface*.

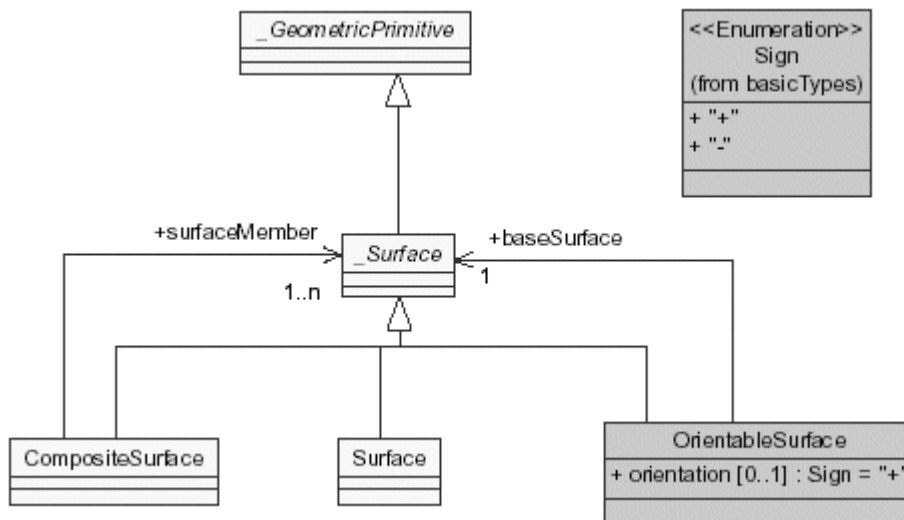


Figure 9: gml3.0 definition for orientable and composite surface

An *OrientableSurface* (type *OrientableSurfaceType*) consists of a surface and an orientation. If the orientation is "+", then the orientable surface is identical to the base surface. If the orientation is "-", then it is reversed.

There is no equivalence in IFC2x(2), any surface has its orientation, and the base definition would have to be used to change the orientation (e.g. by rotating the surface normal).

*Note: There is an entity oriented\_surface in Part42 3<sup>rd</sup> Ed. It could be used later in IFC, but needs to be adjusted as it uses ANDOR based inheritance.*

A *CompositeSurface* (type *CompositeSurfaceType*) is geometry type with all the geometric properties of a (primitive) surface. Essentially, a composite surface is a collection of surfaces that join in pairs on common boundary curves and which, when considered as a whole, form a single surface.

There is no equivalence in IFC2x(2) as a pure geometry item.

However the *IfcConnectedFaceSet* (as a topology item) would have the same capabilities, when using a geometric definition for *IfcFace*, like *IfcFaceSurface* or *IfcPolyLoop*.

If many connected face sets are to be joint as a composite, the *IfcFaceBasedSurfaceModel* (as a geometric item) would be most appropriate.

## 2.5 Geometric Primitives (3-dimensional)

The 3 dimensional geometrical primitives and items include solids.

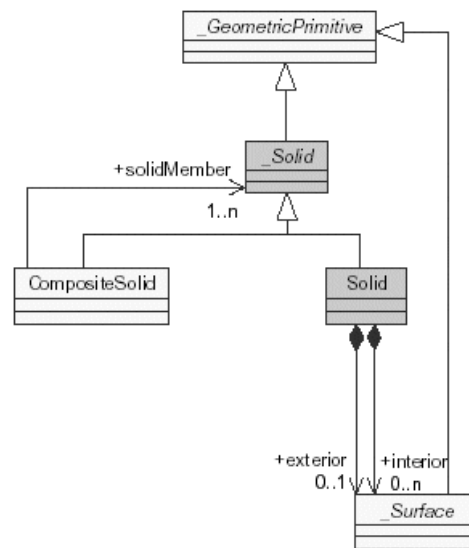


Figure 10: gml3.0 definition of solid

A *\_Solid* (type *AbstractSolidType*) is an abstraction of a solid to support the different levels of complexity. A solid is always contiguous. It is an abstract element and type and the head of the substitution group.

There are several property types, such as *SolidPropertyType*, *solidProperty*, *SolidArrayPropertyType*, *SolidArrayProperty* used to reference or include solids as a property.

A *Solid* (type *SolidType*) is the non-abstract element for 3 dimensional solids.

The extent of a solid is defined by the boundary surfaces (shells). A shell is represented by a composite surface, where every shell is used to represent a single connected component of the boundary of a solid. It consists of a composite surface (a list of orientable surfaces) connected in a topological cycle (an object whose boundary is empty).

An *CompositeSolid* (type *CompositeSolidType*) a collection of solids that join in pairs on common boundary surfaces and which, when considered as a whole, form a single solid.

The equivalence in IFC2x(2) is the *IfcSolidModel*. It is the abstract supertype of all solid models.

*Note: There are other 3D models, then solid models, in IFC.*

The solid definition refers to a boundary representation, or BREP, and not to an arbitrary solid. The equivalent in IFC2x(2) is the *IfcManifoldSolidBrep*.

It is an abstract type having two subtypes. *IfcFacetedBrep* and *IfcFacetedBrepWithVoids*. The *Solid* definition allows for voids.

There is no direct equivalence in IFC2x(2).

The *IfcShellBasedSurfaceModel* may be used as it describes the collection of many shells, including closed shells. However it does not demand shells to be solids and can therefore only be used as a work around.

## 2.6 Geometric aggregates

The geometric aggregates seem to be used to define collections of geometric items for some form of shape representation. In IFC2x(2) there are no direct equivalencies to these definitions.

However the IFC geometry, following the structure and methodology of ISO10303, differentiates between the geometric representation items and their collection to a shape representation. These shape representations can then be classified to be either line based, surface based, solid based (or even with more detail).

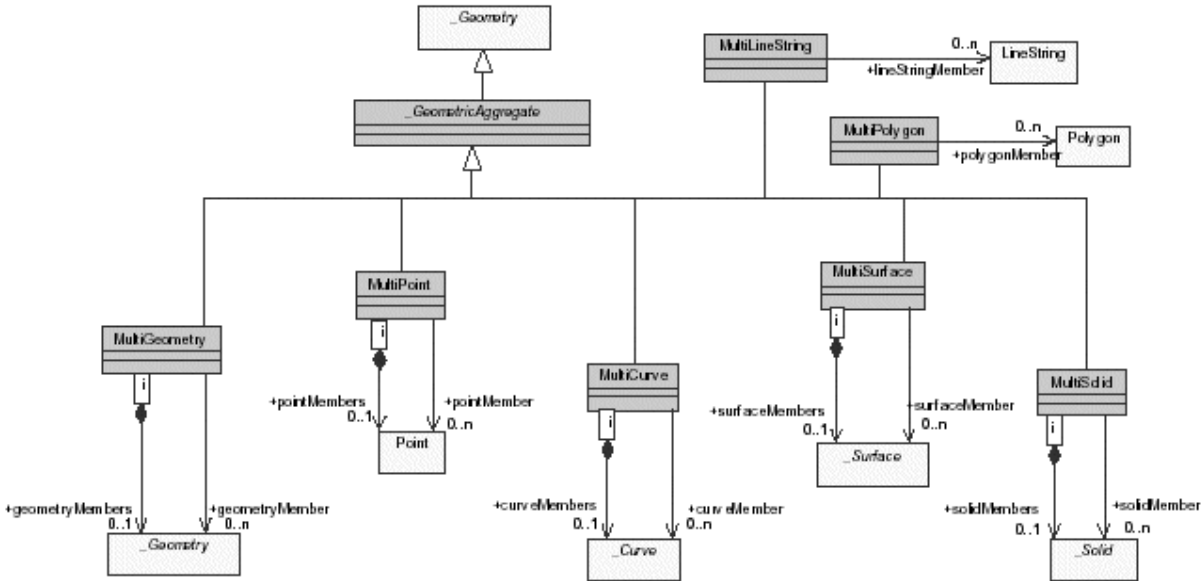


Figure 11: gml3.0 definitions of geometric aggregates

The *MultiGeometry* allows for an arbitrary collection of any geometry, the other elements, *MultiPoint*, *MultiCurve*, *MultiLineString*, *MultiPolygon*, *MultiSurface*, and *MultiSolid*, restrict the aggregate to a specific type of geometry. Each geometric aggregate can be defined by referencing a single element using the array structure (like *geometryMembers*), or by many elements, referencing a single geometry element (like *geometryMember*).

In IFC2x(2) the *IfcShapeRepresentation* has one to many *Items*, each being any subtype of *IfcGeometricRepresentationItem*. The attribute *RepresentationType* is used to add constraints on which type of geometric representation items is allowed for a valid instance of *IfcShapeRepresentation*.

Note: Within ISO 10303-43 and the application protocols using this integrated resource special subtypes of *shape\_representation* are introduced.

The current IFC2x2 release includes the following:

Curve2D	2 dimensional curves
GeometricSet	points, curves, surfaces (2 or 3 dimensional)
GeometricCurveSet	points, curves (2 or 3 dimensional)
SurfaceModel	face based and shell based surface model
SolidModel	including swept solid, Boolean results and Brep bodies
SweptSolid	swept area solids, by extrusion and revolution
Brep	faceted Brep's with and without voids
CSG	Boolean results of operations between solid models, half spaces and Boolean results
Clipping	Boolean differences between swept area solids, half spaces and Boolean results
AdvancedSweptSolid	swept area solids created by sweeping a profile along a directrix
BoundingBox	simplistic 3D representation by a bounding box
SectionedSpine	cross section based representation of a spine curve and planar cross sections. It can represent a surface or a solid and the interpolations of the between the cross sections is not defined
MappedRepresentation	representation based on a mapped item, referring to a representation map. Note: it can be seen as an inserted block reference. The shape representation of the mapped item has a representation type declaring the type of its representation items.

Table 1: List of shape representation types in IFC

## 2.7 Geometric properties

The gml3.0 defines geometric properties, that associates instances of these geometry types with features. A feature can have several geometric properties (to be defined and restricted in application schemas). Each geometric property has a formal name given by its XSD element name.

Formal name	Geometry type
<i>pointProperty</i>	<i>Point</i>
<i>curveProperty</i>	<i>LineString</i>
	<i>Curve</i>
	<i>OrientableCurve</i>
	<i>CompositeCurve</i>
<i>surfaceProperty</i>	<i>Polygon</i>
	<i>Surface</i>
	<i>OrientableSurface</i>
	<i>CompositeSurface</i>
<i>solidProperty</i>	<i>Solid</i>
	<i>CompositeSolid</i>
<i>geometryProperty</i>	<i>AbstractGeometry</i>
<i>multiPointProperty</i>	<i>MultiPoint</i>
<i>multiCurveProperty</i>	<i>MultiCurve</i>
<i>multiSurfaceProperty</i>	<i>MultiSurface</i>
<i>multiSolidProperty</i>	<i>MultiSolid</i>
<i>multiGeometryProperty</i>	<i>MultiGeometry</i>
<i>pointArrayProperty</i>	<i>Point(s)</i>
<i>curveArrayProperty</i>	<i>LineString(s)</i>
	<i>Curve(s)</i>
	<i>OrientableCurve(s)</i>
	<i>CompositeCurve(s)</i>

Formal name	Geometry type
<i>surfaceArrayProperty</i>	<i>Polygon(s)</i>
	<i>Surface(s)</i>
	<i>OrientableSurface(s)</i>
	<i>CompositeSurface(s)</i>
<i>solidArrayProperty</i>	<i>Solid(s)</i>
	<i>CompositeSolid(s)</i>

Table 2: Geometric properties in gml3.0

In IFC2x(2) geometric properties are attached to products (the IFC equivalence to features) through the *IfcProductDefinitionShape*, referencing one or many *IfcShapeRepresentation*. Each of the shape representation has a defined type (see Table 1) and an *RepresentationIdentifier*. The identifier provides a descriptive name of what the shape representation of this product means. Examples are Axis, FootPrint, Body, etc.

## 2.8 User-defined Geometry Types and Geometry Property Types

Since gml3.0 is a meta schema, that can be extended in application schemas, it offers the possibility to application schema developers to introduce their own geometric types as extensions of the gml3.0 base geometry types and geometry property types.

The IFC model is both, it has an “built-in” meta schema, but it also provides all application specific extensions (like the domain schemas). The IFC model is not meant to be extended within independent application schemas. Therefore the provision of user defined geometry types is out of scope of IFC.

### 3 Coordinate Reference Systems

The coordinate reference system plays a key role for all GIS systems.

The geo referencing of the CAD world within a GIS system is important and needs to be provided. Before comparing the gml3.0 definitions the meaning and definition of coordinate reference system should be discussed.

#### 3.1 Meaning of the coordinate reference system

The following explanations are quoted from the gml3.0 specification, page 122:

*The primary object to be referenced is a CRS. The method of referencing is specified in other clauses of this document which describe the various objects and their properties. The CRS instance will give information about the coordinates, including the order of the coordinates, the unit of measure that goes with each coordinate, and the physical meaning in terms of its attachment to the earth (that is, the datum). Also, it is sometimes necessary to express a conversion or transformation to another CRS. Definitions of coordinate conversions and transformations are also encoded using the CRS schemas.*

*Most CRSs used with GML will be references to standard or well-known coordinate reference systems. The XML schemas allow information for a set of standard CRSs to be stored in a dictionary. It is then necessary only to reference the proper CRS in such a dictionary, which will contain all the information needed to understand the coordinates. It is also possible to convert coordinate data into another CRS, using a coordinate transformation service. There is already an OGC Implementation Specification for such a service, document 01-009, and available implementations of this service. Using this service, a set of coordinates in one CRS can be converted to another CRS.*

*Another possible use is when coordinates are not in a standard coordinate reference system. In this case, it is necessary for the XML document to reference a non-standard CRS, which may be defined within the same document. The definition may be as simple as giving the meaning of the axes (order and units of measure) and the datum, or it may include a transformation or conversion to a standard CRS. Thus, the CRS schemas allow a XML document to specify the meaning of its coordinates, and their relationship to a standard, earth-related coordinate system.*

#### 3.2 Definition of coordinate reference systems

The openGIS Abstract Specification, Topic 2: Spatial referencing by coordinates differentiates between different types of CRSs. The following principal sub-types of coordinate reference system are distinguished:

- a) **Geocentric.** Type of coordinate reference system that deals with the earth's curvature by taking the 3D spatial view, which obviates the need to model the earth's curvature. The origin of a geocentric CRS is at the approximate centre of mass of the earth.
- b) **Geographic.** Type of coordinate reference system based on an ellipsoidal approximation of the geoid. This provides an accurate representation of the geometry of geographic features for a large portion of the earth's surface. Geographic coordinate reference systems can be 2D or 3D. A 2D Geographic CRS is used when positions of features are described on the surface of the reference ellipsoid; a 3D Geographic CRS is used when positions are described on, above or below the reference ellipsoid.
- c) **Projected.** Type of coordinate reference system that is based on an approximation of the shape of the earth's surface by a plane. The distortion that is inherent to the approximation is carefully controlled and known. Distortion correction is commonly applied to calculated bearings and distances to produce values that are a close match to actual field values.
- d) **Engineering.** Type of coordinate reference system that is that is used only in a contextually local sense. This sub-type is used to model two broad categories of local coordinate reference systems:

- earth-fixed systems, applied to engineering activities on or near the surface of the earth;
- coordinates on moving platforms such as road vehicles, vessels, aircraft or spacecraft<sup>1</sup>.

Earth-fixed Engineering CRSs are commonly based on a simple flat-earth approximation of the earth's surface, and the effect of earth curvature on feature geometry is ignored: calculations on coordinates use simple plane arithmetic without any corrections for earth curvature. The application of such Engineering CRSs to relatively small areas and "contextually local" is in this case equivalent to "spatially local".

- e) **Vertical.** Type of coordinate reference system used for the recording of heights or depths. Vertical CRSs make use of the direction of gravity to define the concept of height or depth, but its relationship with gravity may not be straightforward. By implication ellipsoidal heights (h) cannot be captured in a vertical coordinate reference system. Ellipsoidal heights cannot exist independently, but only as inseparable part of a 3D coordinate tuple defined in a geographic 3D coordinate reference system.

The "word coordinate system" of the CAD presentation of a building plan is an example of an earth-fixed engineering coordinate reference system.

In addition also Image and Temporal coordinate systems are defined. However these are out of scope of this work.

Two additional subtypes of coordinate reference systems need to be distinguished, compound coordinate reference systems and derived coordinate reference systems. A common example for compound CRS is a combination of a geographic 2D + vertical.

### 3.2.1 Definition of coordinate systems

Each coordinate reference system uses an coordinate system. Compound coordinate reference systems may use different coordinate systems for the parts that make up the compound CRS.

The coordinates of points are recorded in a coordinate system. A coordinate system is the set of coordinate system axes that spans the coordinate space. A coordinate system can have one to many axes, typically these are:

- 1D – e.g. a linear or vertical coordinate system to measure heights along a single axis
- 2D – e.g. the coordinate system of a 2D map or 2D building plan
- 3D – e.g. the coordinate system of a 3D terrain model or building model
- 4D – e.g. adding the temporal axis to a 3D model

The following types are distinguished in openGIS Abstract Specification, Topic 2: Spatial referencing by coordinates differentiates between different types of CRSs.

CS subtype	Description	Used with CRS type(s)
Cartesian	1-, 2-, or 3-dimensional coordinate system. It gives the position of points relative to orthogonal straight axes in the 2- and 3-dimensional cases. In the 1-dimensional case, it contains a single straight coordinate axis. In the multi-dimensional case, all axes shall have the same unit of measure.	Geocentric Projected Engineering Image
oblique Cartesian	2- or 3-dimensional coordinate system with straight axes that are not necessarily orthogonal.	Engineering Image
ellipsoidal	2- or 3-dimensional coordinate system in which position is specified by geodetic latitude, geodetic longitude and (in the three-dimensional case) ellipsoidal height, associated with one or more geographic coordinate reference systems.	Geographic Engineering

<sup>1</sup> Out of scope for this work.

CS subtype	Description	Used with CRS type(s)
spherical	3-dimensional coordinate system with one distance, measured from the origin, and two angular coordinates. Not to be confused with an ellipsoidal coordinate system based on an ellipsoid 'degenerated' into a sphere	Geocentric Engineering
cylindrical	3-dimensional coordinate system consisting of a polar coordinate system extended by a straight coordinate axis perpendicular to the plane spanned by the polar coordinate system.	Engineering
polar	2-dimensional coordinate system in which position is specified by distance to the origin and the angle between the line from origin to point and a reference direction.	Engineering
vertical	1-dimensional coordinate system used to record the heights (or depths) of points dependent on the Earth's gravity field. An exact definition is deliberately not provided as the complexities of the subject fall outside the scope of this specification.	Vertical Engineering
linear	1-dimensional coordinate system that consists of the points that lie on the single axis described. The associated ordinate is the distance from the specified origin to the point along the axis. Example: usage of the line feature representing a road to describe points on or along that road.	Engineering
temporal	1-dimensional coordinate system containing a single time axis and used to describe the temporal position of a point in the specified time units from a specified time origin.	Temporal

Typically building projects are designed in an Cartesian coordinate system within an engineering coordinate reference system, hence:

CS subtype	Description	Used with CRS type(s)
Cartesian	2-, or 3-dimensional coordinate system. It gives the position of points relative to orthogonal straight axes in the 2- and 3-dimensional cases. All axes shall have the same unit of measure.	Engineering (no datum)

### 3.2.2 Definition of coordinate axes

Coordinate systems are composed of an ordered set of coordinate system axes. An axes can be further distinguished by the way it is defined, e.g. by distance from a origin (given by a length measure), or by an planar angle from a reference direction (given by a planar angle measure) or along a time axis.

CS	CRS	Permitted coordinate system axis names
Cartesian	Geocentric	Geocentric X, Geocentric Y, Geocentric Z
Spherical	Geocentric	Spherical Latitude, Spherical Longitude, Geocentric Radius
Ellipsoidal	Geographic	Geodetic Latitude, Geodetic Longitude, Ellipsoidal height (if 3D)
Vertical	Vertical	Gravity-related height
Vertical	Vertical	Depth
Cartesian	Projected	Easting, Northing
Cartesian	Projected	Westing, Southing

Typically building projects are designed in an Cartesian coordinate system with axis definitions right handed, X, Y, Z. There is no datum, all axis refer to an (arbitrary) location of origin (0.,0.,0.). The single underlying Cartesian coordinate system of a CAD model used for building plans is also called “World Coordinate System”, or WCS.

Often a CAD model of a building plan has many local coordinate systems, that all are translated by a transformation matrix (3x3 for 2D or 4x4 for 3D) into the single “World Coordinate System”. This local coordinate systems are named “user coordinate system”, UCS, or “object coordinate system”, OCS – all are Cartesian, right-handed, 2D or 3D coordinate systems.

Hence the axes are named:

CS	CRS	Permitted coordinate system axis names
Cartesian	Engineering	(global) X, (global) Y, (global) Z ; local X, local Y, local Z

### 3.2.3 Definition of datum

A datum specifies the relationship of a coordinate system to the earth and is needed to create a coordinate reference system. In cases of engineering coordinate system it can also be defined locally and not in the relationship to the earth.

There are different types of datum, each type can only be assigned to a specific type of coordinate reference system. These are:

- geodetic datum
- vertical datum
- image datum
- engineering datum
- temporal datum

Each of them can be further classified, a vertical datum can be geoidal, based on normal sea level, etc. A geodetic datum is often associated with a geographic coordinate reference system and an ellipsoidal coordinate system.

### 3.3 GML definition of coordinate reference systems

GML requires a coordinate reference system (CRS) to be referenced whenever location coordinate information is given. This CRS provides the meaning for location coordinates. The referencing is generally given using the *srsName* attribute which is provided by *AbstractGeometryType* which is the basis for the content models for all GML geometry elements.

Therefore each geometry element that provides location information needs to carry the *srsName* attribute. For examples a *Point* `<gml:Point srsName="utm27n"> . . .`

For IFC (as for all CAD systems) the coordinate reference system is always a notional world coordinate system given by a Cartesian coordinate system. As all geometry refers to a Cartesian coordinate system there is no need to reference a coordinate reference system.

A *srsName* points to a CRS instance. For well known references it is not required that the CRS description exists at the location the URI points to. It can be given by a significant name string that uniquely defined the CRS.

The CRS determines the interpretation of the location coordinates as coordinates along the coordinate axis system of the CRS.

There is no equivalent in IFC2x(2). All coordinates are given (directly or indirectly) within the world coordinate system, established by the *IfcGeometricRepresentationContext*. It establishes an:

- Engineering CRS with
- Cartesian CS having 2 or 3
- Cartesian Coordinate Axes (X, Y, (Z))

The *srsName* can determine the CRS by name or by explicit definition. The explicit model to define CRS is currently not fully released by the OGS, only an OpenGIS Recommendation Paper exists. It defines:

- Reference systems
- Coordinate reference systems
- Coordinate systems
- Coordinate system axes
- Datum
- Coordinate operations, conversions and operations
- Data quality

The only informational reference to a geographic coordinate reference system that currently exists in IFC2x(2) is given at *IfcSite*. *IfcSite* definition currently includes a reference longitude, latitude and elevation (see Figure 12). However the original use case to include these references has not been to allow for geo-referencing, but to include a geographic location to enable the selection of climate data from sources close to the location (like the typical climatic year for energy simulations). Therefore these references are not directly related to a coordinate within the engineering CRS of the project and the coordinate values are restricted to be degree, minute, second for longitude and latitude (based on a geographic coordinate system with Greenwich as prime meridian and the equator) and a length unit for the elevation as being relative to the average sea level (without explicitly stating the datum of the elevation – or which sea level is meant). For the current use case this accuracy is fully appropriate, but it would not be sufficient for geo-referencing.

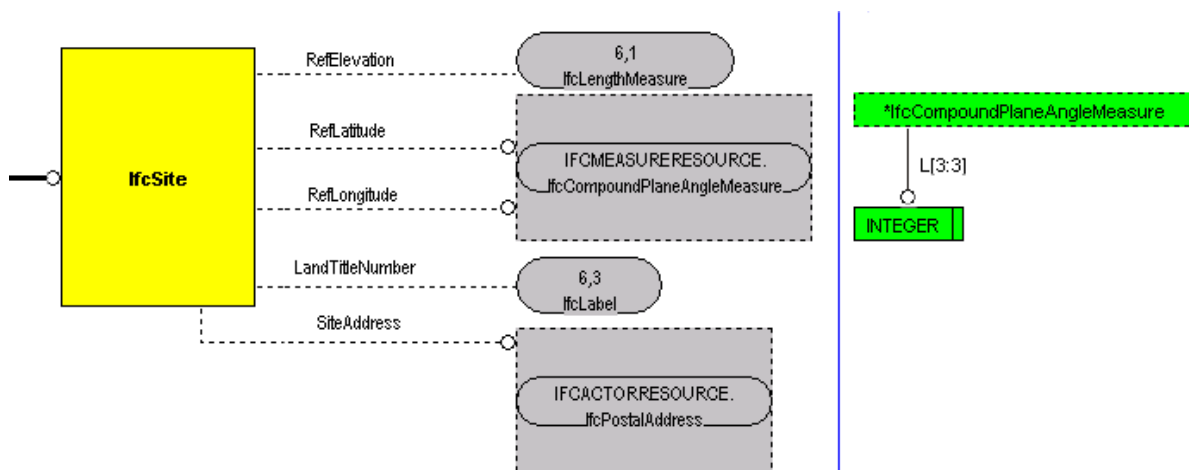


Figure 12: references to geographic coordinates at *IfcSite*

A full definition of CRS within IFC is certainly not appropriate. IFC should (as the CAD systems delivering IFC compliant data) be restricted to an engineering CRS, based on a Cartesian (right-handed) 2D and 3D coordinate system.

However IFC should allow a proper geo-referencing. The world coordinate system, as defined in IFC by *IfcGeometricRepresentationContext*, should be enabled to carry its position within another reference coordinate system, such as a geocentric, geographic or projected coordinate system. The reference should contain:

- the significant name of the CRS – as recommended by OGS for *srsName*
- implicitly the CRS and CS associated by the *srsName*
- the coordinates as either Cartesian, Spherical, Ellipsoidal, or Vertical depending on the CS
- the datum

This reference would relate the origin and the reference axes (x, y, z) of the engineering CRS (the world coordinate system of CAD) to the underlying CRS of the GIS source.

Another possibility is storing corresponding coordinates (in the engineering CRS of the building plan and the CRS of the GIS source) for significant points within the terrain handed over to the CAD system. Examples are the control point or trigonometric points which are known by their position according to a datum.

## 4 Comparison of Topology schema

Both gml3.0 and IFC2x(2) have a topology schema. The IFC topology is based on the ISO 10303-42 standard. Topological objects exist for 0D (vertex), 1D (edge), 2D (face), and 3D (shell). The gml3.0 definition of topology is based on the so-called “relational topology” concept, that objectifies the boundary relationships between the topological primitives.

The high level gml3.0 breakdown of topology objects is described in .

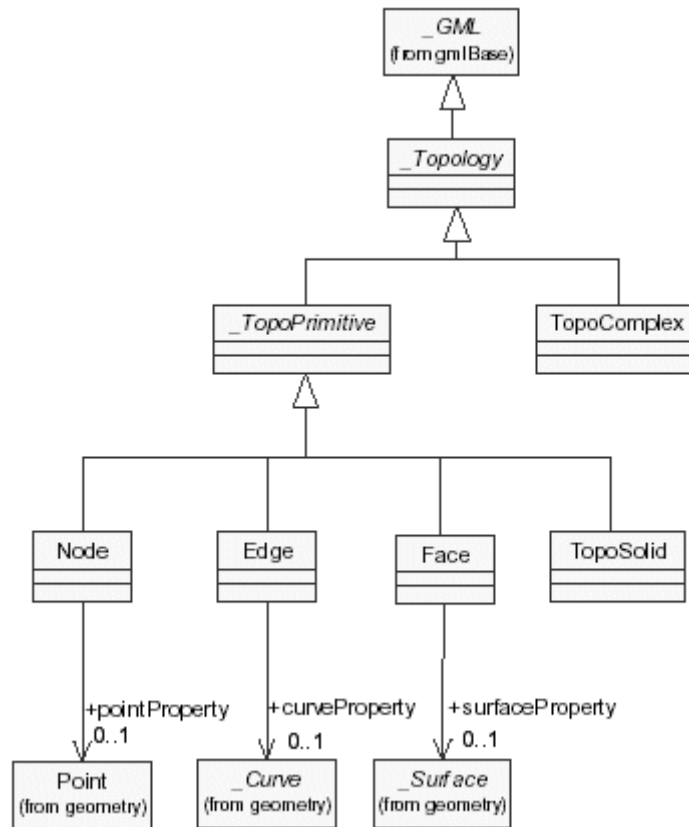


Figure 13: GML definition of topology

The topological primitives are node, edge, face and a topological solid (or shell ?). They also include the possibility to use geometric items (point, curve, surface) to provide a geometric shape. The topological items have boundary relations to each other.

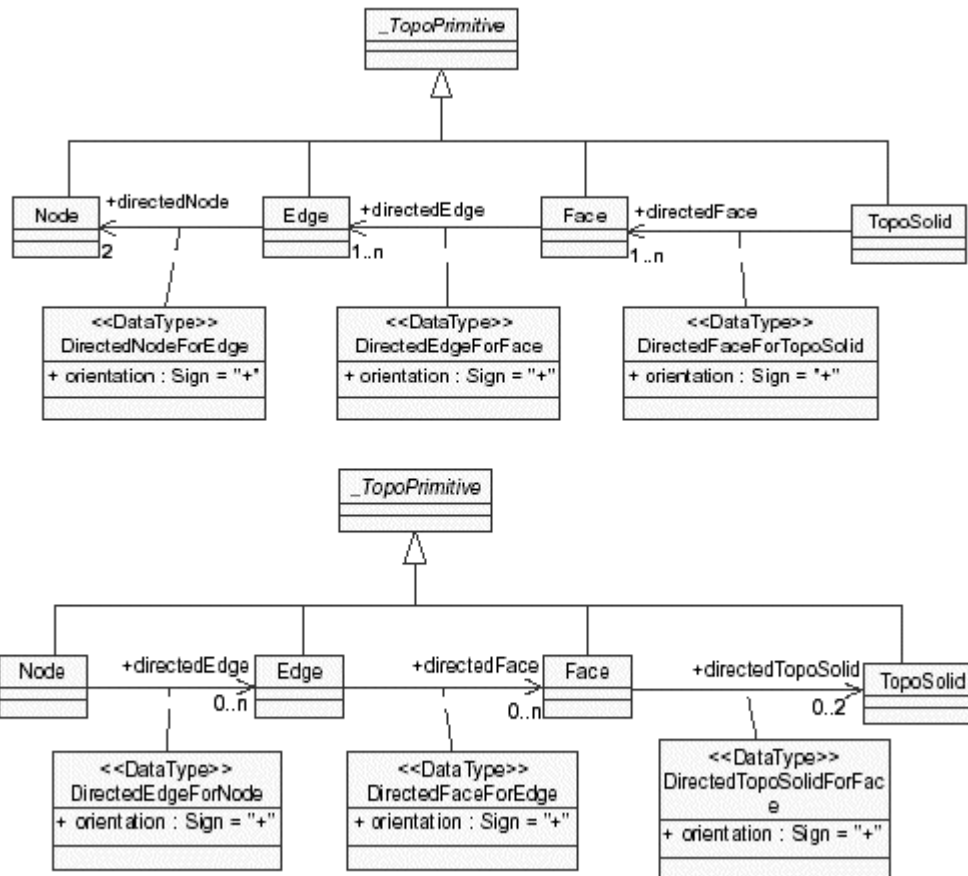


Figure 14: GML definition of topological boundary conditions

The next table compares these primitives:

A *\_Topology* (type *AbstractTopologyType*) supplies the root for all topological elements.

A *\_TopoPrimitive* (type *AbstractTopoPrimitiveType*) is the base type for topological primitive elements. It has optional references to isolated topological items (a face may isolate an edge) and optional references to container items (a node may have faces as containers)

A *\_Node* (type *NodeType*) represents the 0-dimensional primitive expressing point coincidence. The topological boundary of a node is empty and hence requires no representation. The optional co-boundary of a node is a set of directed edges which are incident on this node.

The *NodeType* includes a reference to the element *pointProperty*. It enables the use of a geometric point entity to describe the position of the node.

The equivalence in IFC2x(2) is the abstract *IfcTopologicalRepresentationItem*.

There is no equivalence in IFC2x(2). The IFC topology (based on ISO 10303-42) does not distinguish by a first level subtype hierarchy between topological primitives and composites.

The equivalence in IFC2x(2) is the *IfcVertex*. A vertex is the topological construct corresponding to a point. It has dimensionality 0 and extent 0.

In IFC2x(2) it is described by the *IfcVertexPoint*. It is a subtype of *IfcVertex* with the additional reference to an *IfcPoint*.

The *directedNode* (type *DirectedNodePropertyType*) adds an orientation to the node and is used as a role in the boundary relation to edges.

A *\_Edge* (type *EdgeType*) represent the 1-dimensional primitive expressing linear coincidence. The topological boundary of an Edge consists of a negatively directed start Node and a positively directed end Node. The optional co-boundary of an edge is a circular sequence of directed faces which are incident on this edge.

The boundary relation is given by reference to *directedNode*, the co-boundary by reference to *directedFace*.

The *EdgeType* includes a reference to the element *curveProperty*. It enables the use of a geometric curve entity to describe the exact geometric domain of the edge.

The *directedEdge* (type *DirectedEdgePropertyType*) adds an orientation to the edge and is used as a role in the boundary relation to edges.

A *\_Face* (type *FaceType*) represent the 2-dimensional topology primitive expressing surface overlap. The topological boundary of a face consists of a set of directed edges.

The non-dangling edges in the boundary of a face comprise one or more topological rings. Each such ring consists of *directedEdges* connected in a cycle, and is oriented with the face on its left. The optional co-boundary of a face is a pair of directed solids which are bounded by this face.

GML makes a difference between dangling and non-dangling edges (dangling edges have the same face on both sides). IFC does not make such a differentiation.

The *FaceType* includes a reference to the element *surfaceProperty*. It enables the use of a geometric surface entity to describe the exact geometric domain of the face.

There is no equivalence in IFC2x(2) since the boundary relations are not objectified (see also ISO 10303-42).

The equivalence in IFC2x(2) is *lfcEdge*. An edge is the topological construct corresponding to the connection of two vertices.

The topological boundary is given by the direct relations *EdgeStart* and *EdgeEnd* to *lfcVertex*.

Co-boundaries are not described in IFC (based on ISO 10303-42).

In IFC2x(2) it is described by the *lfcEdgeCurve*. It is a subtype of *lfcEdge* with the additional reference to an *lfcCurve*.

There is no direct equivalence in IFC2x(2) since the boundary relations are not objectified (see also ISO 10303-42).

However there is an *lfcOrientedEdge* that adds an orientation and enables to switch between start and end vertex. It enables the use of a “traversal” of the edge in solid modeling.

The equivalence in IFC2x(2) is *lfcFace*. A face is a topological entity of dimensionality 2 corresponding to the intuitive notion of a piece of surface bounded by loops.

The topological boundary is given by the direct relation to one-to-many *lfcFaceBound*. Exactly one of them should be an *lfcFaceOuterBound*. Bounds can be given by either *lfcEdgeLoop*, an ordered collection of *lfcOrientedEdge* (corresponds to the idea of a ring in GML). It can also be given by an *lfcVertexLoop* consisting of a single vertex (this concept seems to be not included in GML).

To support a shortcut for faceted representations a bound can also be given by *lfcPolyLoop*, being an ordered collection of Cartesian points (and thereby providing edges only implicitly as straight edges). This concept is not included in GML.

Co-boundaries are not described in IFC (based on ISO 10303-42).

In IFC2x(2) it is described by the *lfcFaceSurface*. It is a subtype of *lfcFace* with the additional reference to an *lfcSurface*.

The *directedFace* (type *DirectedFacePropertyType*) adds an orientation to the face and is used as a role in the boundary relation to faces.

A *TopoSolid* (type *TopoSolidType*) represent the 3-dimensional topology primitive expressing Volume interclause. The topological boundary of a *TopoSolid* consists of a set of directed faces.

The *directedTopoSolid* (type *DirectedTopoSolidPropertyType*) adds an orientation to the topological solid and is used as a role in the co-boundary relation to faces.

There is no direct equivalence in IFC2x(2) since the boundary relations are not objectified (see also ISO 10303-42).

However the *IfcFaceBound*, used to bound the *IfcFace* has an attribute Orientation.

The equivalence in IFC2x(2) is *IfcShell*. A closed shell is a shell of the dimensionality 2 which typically serves as a bound for a region in R3. A closed shell has no boundary, and has non-zero finite extent.

If the shell is to be used to describe a solid, i.e. a geometry model, the *IfcShell* has to be used within the container of either *IfcFacetedBrep* (to represent a true volume), or (for a surface based model) *IfcShellBasedSurfaceModel*.

There is no direct equivalence in IFC2x(2) since the boundary relations are not objectified (see also ISO 10303-42).

In additions to the topological items introduced above, gml3.0 also defines topological items to be used to represent point, curve, surface or volume features. For each of those items an additional element defining them as a property feature is given. This refers to:

- *TopoPoint, TopoPointProperty*
- *TopoCurve, TopoCurveProperty*
- *TopoSurface, TopoSurfaceProperty*
- *TopoVolume, TopoVolumeProperty*

There is no direct equivalence to these elements within IFC2x(2), the use of topological and/or geometric representation items to represent products (the “features” in IFC) is handled via the *IfcProductRepresentation* and *IfcShapeRepresentation / IfcTopologyRepresentation*.

The gml3.0 specification also contains a topology complex, the *TopoComplex*. It can comprise multiple topology primitive members. There is no equivalence in IFC2x(2), but there are geometric models defined in IFC that can comprise multiple topological elements and thereby implicitly defining a collection of topology:

- *IfcFaceBasedSurfaceModel*, including 1:N connected face sets, each containing 1:N topological faces
- *IfcShellBasedSurfaceModel*, including 1:N shells (closed and open shells), each containing 1:N topological faces (and in case of closed shell also 1:N topologically bounded volumes)
- *IfcManifoldSolidBrep*, including 1 closed shell as a topologically bounded volume.

# Appendix 1 Terms and Definitions

## Appendix 1.1 Definition of coordinate related terms

A coordinate reference system is a coordinate system that has a reference to the Earth. Coordinates are unambiguous only when the coordinate reference system to which those coordinates are related has been fully defined. That is the purpose of the coordinate reference system CRS.

A few definitions<sup>2</sup>:

### Cartesian coordinate system

*Coordinate system which gives the position of points relative to N mutually-perpendicular straight axes. In the context of geospatial coordinates the maximum value of N is three.*

### Coordinate

*one of a sequence of N numbers designating the position of a point in N-dimensional space. In a coordinate reference system, the coordinate numbers must be qualified by units.*

### Coordinate reference system, CRS

*Coordinate system which is related to the real world by a datum. For geodetic and vertical datums, it will be related to the Earth.*

### Coordinate system, CS

*Set of (mathematical) rules for specifying how coordinates are to be assigned to points*

*NOTE 1 One coordinate system may be used in many coordinate reference systems.*

*NOTE 2 The geometric properties of a coordinate space determine how distances and angles between points are calculated from the coordinates. For example, in an ellipsoidal (2D) space distances are defined as curves on the surface of the ellipsoid, whereas in a Euclidean plane as used for projected CRS distance is the length of a straight line between two points. The mathematical rules that determine distances and angles are calculated from coordinates and vice versa are comprised in the concept of coordinate system.*

### Datum

*parameter or set of parameters that determine the location of the origin, the orientation and the scale of a coordinate reference system*

### Geodetic coordinate system

*Coordinate system in which position is specified by geodetic latitude, geodetic longitude and (in the three-dimensional case) ellipsoidal height, associated with one or more geographic coordinate reference systems.*

### Engineering coordinate reference system

*A coordinate reference system that is defined for and usually used in a contextually local sense, which may be an area, significantly less than the complete surface of the earth or a moving platform and its vicinity.<sup>3</sup>*

*EXAMPLE Local engineering and architectural coordinates, grids, and drawings*

*NOTE 1 A transformation of engineering coordinates to geodetic coordinates may or may not be possible depending on whether such operation parameters have been determined (or defined).*

### Geocentric coordinate reference system

---

<sup>2</sup> quoted from The OpenGIS® Abstract Specification, Topic 2: Spatial referencing by coordinates

<sup>3</sup> Editors note: Moving platforms out of scope of this document – only non-moving, permanent engineering coordinate reference systems are in scope.

*3-dimensional coordinate reference system with its origin at the (approximate) centre of the Earth.*

**Geodetic coordinates**

*Coordinates defined in a geocentric, geographic (2D or 3D) or projected coordinate reference system.*

**Geodetic Datum**

*Datum describing the relationship of a 3D or 2D coordinate system to the Earth, In most cases, the geodetic datum includes an ellipsoid definition.*

**Local datum, engineering datum**

*datum with a local reference, used as a basis for an engineering coordinate reference system. Engineering datum excludes both geodetic and vertical datums.*

**Map projection**

*Conversion from a geodetic coordinate system to a planar surface*

**Projected coordinate reference system**

*Two-dimensional coordinate system resulting from a map projection.*

*NOTE 1 A projected coordinate reference system is derived from a 2D geographic coordinate reference system by applying a parameterised coordinate transformation known as a ‘map projection’.*

*NOTE 2 A projected coordinate reference system commonly uses a Cartesian coordinate system.*

**Spherical coordinate system**

*3-dimensional coordinate system with one distance, measured from the origin and two angular coordinates, commonly associated with a geocentric coordinate reference system*