# OPENGIS PROJECT DOCUMENT 13-083r1

| | |
|---|---|
| **TITLE:** | **NWIP ISO 19107 Geographic information-Spatial Schema** |
| **AUTHOR:** | **Name: John R. Herring, Simple Features SWG** |
| **DATE:** | **29 August 2013** |
| **CATEGORY:** | **Proposal** |

## 1. Package description

This is a NWIP (New Work Item Proposal) package intended for submission to ISO TC 211 to kick off a project to update ***ISO 19107:2003 Geographic information – Spatial schema***. It contains a 2 page NWIP proposal and a Draft document from which the project is intended to begin its work. This document can be considered to consist of 3 parts:

1. This cover page
2. The NWIP proposal (the next 2 pages)
3. A new Draft of ISO 19107

The Simple Features SWG has moved to propose to the TC (in Frascati)

- To submit this NWIP with its attached Draft to ISO, and
- To initiate an OGC RFC on the Draft to supply ISO (as a liaison organization) with comments for the use of the ISO Project Team (which must contain OGC representatives).

The RFC will contain only the last part of this document, the new Draft for ISO 19107, which is currently the geometry model in use in OGC since about 1999. All technical aspects in the Draft have been implemented by several member companies of the OGC.

The next draft of the Simple Features standard (also ISO 19125) will leverage from the new 19107, and probably contain:

1. A feature model identical to the one used in the original Simple Features.
2. A geometry model taken from the conformance classes of the new 19107
3. (For the SQL volume) SQL syntax from SQL/MM (ISO/IEC 13249-3:2011 Information technology - Database languages - SQL multimedia and application packages - Part 3: Spatial)
4. And the text extension in SF 1.2

| NEW WORK ITEM PROPOSAL | |
|---|---|
| Closing date for voting | Reference number<br>(to be given by the Secretariat) |
| Date of circulation | **ISO/TC** 211 **/ SC** **N** |
| Secretariat | ☐ **Proposal for new PC** |

A proposal for a new work item within the scope of an existing committee shall be submitted to the secretariat of that committee with a copy to the Central Secretariat and, in the case of a subcommittee, a copy to the secretariat of the parent technical committee. Proposals not within the scope of an existing committee shall be submitted to the secretariat of the ISO Technical Management Board.

The proposer of a new work item may be a member body of ISO, the secretariat itself, another technical committee or subcommittee, or organization in liaison, the Technical Management Board or one of the advisory groups, or the Secretary-General.

The proposal will be circulated to the P-members of the technical committee or subcommittee for voting, and to the O-members for information.

**IMPORTANT NOTE: Proposals without adequate justification risk rejection or referral to originator**.
Guidelines for proposing and justifying a new work item are contained **in Annex C of the ISO/IEC Directives, Part 1.**

☐ The proposer has considered the guidance given in the Annex C during the preparation of the NWIP.

**Proposal** (to be completed by the proposer)

**Title of the proposed deliverable.**
*(in the case of an amendment, revision or a new part of an existing document, show the reference number and current title)*

English title: ISO 19107: Geographic information - Spatial Schema

French title (if available): Information géographique — Schéma spatial

**Scope of the proposed deliverable.**

This International Standard specifies a conceptual schema for describing the spatial characteristics of geographic features, and a set of spatial operations consistent with these schemas. It treats geometry and topology. It defines standard spatial operations for use in access, query, management, processing, and data exchange of geographic information for spatial (geometric and topological) objects. Because of the nature of geographic information, these geometric coordinate spaces will normally have up to 3 spatial dimensions, 1 temporal dimension, and any number of other as needed by the applications. In general, the topological dimension of the spatial projections of the geometric objects will be at most 3.

**Purpose and justification of the proposal***

Replace ISO 19107:2003. This document will incude several technical corrections and extensions that have been in ciruculation infomally over the past years. Futher, the conformance classes will be restructured to allow for more options in implementations while maintaining an acceptable level of backward compatibility.

**The reason for requiring justification statements with approval or disapproval votes is primarily to collect input on market or stakeholder needs, and on market relevance of the proposal, to benefit the development of the proposed ISO standard(s). Any NSB vote in relation to a proposal for new work may result in significant commitments of resources by all parties (NSBs, committee leaders and delegates/experts) or may have significant implications for ISO's relevance in the global community. It is especially important that NSBs consider and express why they vote the way they do. In addition, it is felt that it would be useful for ISO and its committees to have documentation as to why the NSBs feel a proposal has market need and market relevance. Therefore, please ensure that your justifying statements with your approval or disapproval vote convey the reason(s) why your national consensus does or does not support the market need and/or global relevance of the proposal.*

**If a draft is attached to this proposal,:**

Please select from one of the following options (note that if no option is selected, the default will be the first option):

☐ Draft document will be registered as new project in the committee's work programme (stage 20.00)
☒ Draft document can be registered as a Working Draft (WD – stage 20.20)
☐ Draft document can be registered as a Committee Draft (CD – stage 30.00)
☐ Draft document can be registered as a Draft International Standard (DIS – stage 40.00)

**Is this a Management Systems Standard (MSS)?**

☐ Yes ☒ No

NOTE: if Yes, the NWIP along with the Justification study (see Annex SL of the Consolidated ISO Supplement) must be sent to the MSS Task Force secretariat (tmb@iso.org) for approval before the NWIP ballot can be launched.

**Indication(s) of the preferred type or types of deliverable(s) to be produced under the proposal.**

☒ International Standard ☐ Technical Specification ☐ Publicly Available Specification ☐ Technical Report

**Proposed development track** ☐ 1 (24 months) ☐ 2 (36 months - default) ☒ 3 (48 months)

**Known patented items (see ISO/IEC Directives, Part 1 for important guidance)**

☐ **Yes** ☒ **No** **If "Yes", provide full information as annex**

**A statement from the proposer as to how the proposed work may relate to or impact on existing work, especially existing ISO and IEC deliverables. The proposer should explain how the work differs from apparently similar work, or explain how duplication and conflict will be minimized.**

There are several projects that might have been affected by changes in such a fundmental document. One of the purposes of the new requirements/conformance class sturcture adopted in this document is to make sure that conformance options from the new doucment will cover the needs of these existing standards. All new requirments will be confined to extensions of existing practices and thus the new material will not impede current applications conformant with the older version of this document being able to claim a lever of conformance with this document regardless of their implementation strategy.

**A listing of relevant existing documents at the international, regional and national levels.**

ISO 19107:2003 Geographic information — Spatial schema
ISO 19125-1:2004 Geographic information — Simple feature access -- Part 1: Common architecture
ISO 19125-2:2004 Geographic information — Simple feature access -- Part 2: SQL option
ISO/IEC 13249-3:2006(E) - Information technology — SQL Multimedia and Application Packages - Part 3: Spatial

**A simple and concise statement identifying and describing relevant affected stakeholder categories (including small and medium sized enterprises) and how they will each benefit from or be impacted by the proposed deliverable(s)**

**Liaisons:**

**A listing of relevant external international organizations or internal parties (other ISO and/or IEC committees) to be engaged as liaisons in the development of the deliverable(s).**

OGC
JTC1/SC32

**Joint/parallel work:**

**Possible joint/parallel work with:**

☐ IEC (please specify committee ID)
☒ CEN (please specify committee ID) 287
☐ Other (please specify)

**A listing of relevant countries which are not already P-members of the committee.**

**Preparatory work** (at a minimum an outline should be included with the proposal)

☒ A draft is attached ☐ An outline is attached ☐ An existing document to serve as initial basis

The proposer or the proposer's organization is prepared to undertake the preparatory work required ☐ Yes ☐ No

**Proposed Project Leader** (name and e-mail address)

Dr. John R. Herring, john.herring@oracle.com

**Name of the Proposer**
(include contact information)

OGC Carl Reed (TC Chair)
Email: creed@opengeospatial.org
Phone: +1 970 419 8755 Fax: +1 970 407 1101

**Supplementary information relating to the proposal**

☒ This proposal relates to a new ISO document;

☐ This proposal relates to the adoption as an active project of an item currently registered as a Preliminary Work Item;

☐ This proposal relates to the re-establishment of a cancelled project as an active project.

Other:

**Annex(es) are included with this proposal** (give details)

☐

© ISO 2013 – All rights reserved

**ISO TC 211**

**Date: 2013-09-22**

**Date Last Edited: 2013-08-25**

**ISO TC 211/SC /WG**

**Secretariat: NSF**

# Geographic information — Spatial Schema

*Information géographique — Schéma spatial*

**Warning**

**This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.**

**Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.**

Ed: This is a WORKING DRAFT – An editable copy in Word format is available for download here.
Commenters are welcome to take a copy of this document, and use change tracking to make their suggested changes specific. Upon acceptance, each change will be merged with the baseline document.
The UML model in Enterprise Architect Format is available here.

Document type: International Standard

Document subtype:

Document stage: (20) Preparatory

Document language: E

ISO_19107_2012_(E)

**Copyright notice**

This ISO document is a working draft or committee draft and is copyright-protected by ISO. While the reproduction of working drafts or committee drafts in any form for use by participants in the ISO standards development process is permitted without prior permission from ISO, neither this document nor any extract from it may be reproduced, stored or transmitted in any form for any other purpose without prior written permission from ISO.

Requests for permission to reproduce this document for the purpose of selling it should be addressed as shown below or to ISO's member body in the country of the requester:

[Indicate the full address, telephone number, fax number, telex number, and electronic mail address, as appropriate, of the Copyright Manger of the ISO member body responsible for the secretariat of the TC or SC within the framework of which the working document has been prepared.]

Reproduction for sales purposes may be subject to royalty payments or a licensing agreement.

Violators may be prosecuted.


© ISO 2013 – All rights reserved

# Table of Content

Ed: A clause/model package with "Requirements class" in the title will correspond to a "conformance class" in the abstract test suite. Dependencies between «requirementsclass» packages in the UML will mirror dependencies in the conformance classes.

## Contents Page



© ISO 2013 – All rights reserved

© ISO 2013 – All rights reserved

© ISO 2013 – All rights reserved

© ISO 2013 – All rights reserved

# Table of Figures

## Figures

Page



© ISO 2013 – All rights reserved

© ISO 2013 – All rights reserved

# List of Tables

## Tables

Page



© ISO 2013 – All rights reserved

# Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 19107 was prepared by Technical Committee ISO/TC 211, *Geographic Information*.

This second edition cancels and replaces the first edition (ISO 19107: 2003), which has been technically revised and now forms a logical subset of this document.


© ISO 2013 – All rights reserved

## Introduction

This International Standard provides conceptual schemas for describing and manipulating the spatial characteristics of geographic features. Standardization in this area is the cornerstone for other geographic information design, specification and standardization.

A feature is an abstraction of a real world phenomenon; it is a geographic feature if it can be associated with a least one location as a characteristic or attribute. Each real world phenomenon may be associated to many feature representations which may be associated to different and potentially incompatible locations based on the features' semantic interpretation.

'Vector' data consists of geometric and topological primitives used, separately or in combination, to construct expressions of the spatial characteristics of geographic features. 'Raster' data is based on the division of the extent covered into small units according to a tessellation of the space and the assignment to each unit, or set of units, to a set of attribute values. Raster data can also be used for the expression of spatial characteristics. This International Standard deals only with vector data.

In the model defined in this International Standard, spatial characteristics are described by one or more spatial attributes with each value given by a geometric object (GeometryObject) or a topological object (TopologyObject).

Geometry expressions provide quantitative description, by means of coordinates, mathematical functions and geometric constructions, including dimension, position, size, shape, and orientation. The mathematical functions used for describing the geometry of an object depend on the type of coordinate system used to define the spatial position. Many such functions will be algebraic expressions, but, especially in the case of more complex geometric constructions, closed-form algebraic formulae may not exist or may not be known in the general case. In such cases, numeric approximation may be used with accuracy metadata available if possible.

Topology expressions provide qualitative descriptions of the spatial relations between geometry objects and expressions. Topology deals with the characteristics of geometric figures that remain invariant if the space is deformed elastically and continuously. If done properly, topological detail does not change when information is transformed from one coordinate system to another. Within the context of geographic information, topology is commonly used to describe the connectivity of an *n*-dimensional graph, which is invariant under continuous transformation of the graph. Computational topology provides information about the connectivity of geometric primitives that may be derived from the underlying geometry. In some useful cases, the geometric detail may not be of any consequence to the application of the topological data to problems and may be omitted.

Spatial operators are functions and procedures that use, query, create, modify, or delete spatial objects. This International Standard defines the taxonomy of some of the more important operators, their definitions and implementations. The goals are to:

- Define spatial operators unambiguously, so that diverse implementations can be assured to yield comparable results within limitations of accuracy and resolution.
- Use these definitions to define a set of standard operations that will form the basis of compliant systems, and, thus act as a test-bed for implementers and a benchmark set for validation of compliance.
- Define an operator algebra that will allow combinations of the base operators to be used predictably in the query and manipulation of geographic data.

© ISO 2013 – All rights reserved

Standardized conceptual schemas for spatial characteristics will increase the ability to share geographic information among applications. These schemas will be used by geographic information system and software developers and users of geographic information to provide consistently understandable spatial data structures and functions.


© ISO 2013 – All rights reserved

# Geographic information — Spatial Schema

## 1 Scope

This International Standard specifies a conceptual schema for describing the spatial characteristics of geographic features, and a set of spatial operations consistent with these schemas. It treats geometry and topology. It defines standard spatial operations for use in access, query, management, processing, and data exchange of geographic information for spatial (geometric and topological) objects. Because of the nature of geographic information, these geometric coordinate spaces will normally have up to 3 spatial dimensions, 1 temporal dimension, and any number of other as needed by the applications. In general, the topological dimension of the spatial projections of the geometric objects will be at most 3.

## 2 Conformance

### 2.1 Semantics

This International Standard uses the Unified Modeling Language (UML) to present conceptual schemas for describing the spatial characteristics of geographic features. This schema defines conceptual interfaces that shall be realized in application schemas, profiles and implementation specifications. The document concerns ONLY externally visible interfaces and places no restriction on the underlying implementations other than what is needed to satisfy the interface specifications in the actual situation such as:

1. Interfaces to software libraries and services
2. Interfaces to databases using Query languages such as SQL
3. Data interchange using encoding as defined in ISO 19118.

Few applications will require the full range of capabilities described by this conceptual schema. This international standard, therefore, defines a set of conformance classes that will support applications whose requirements range from the minimum necessary to define data structures to full object implementation. The conformance classes are arranged as a core set of functionality with an orderly progression of extensions, so that a partial implementation of this standard has a logical path to the fully realized functionality.

This flexibility is controlled by a set of UML interface classifiers that can be implemented in a variety of manners. Implementations that define full object functionality must implement all operations and support logical access to the conceptual attributes and associations defined by the classifiers of the chosen conformance class extensions. Implementations that choose to depend on external “free functions” for some or all operations, or forgo them altogether, need not support all operation, but shall always support a data type sufficient to record the state of each of the chosen UML classifier as defined by its member variables and operations. Common names for “metaphorically identical” but technically different entities are acceptable. The UML model in this International Standard defines abstract interfaces, application schemas define conceptual classes, various software systems define implementation classes or data structures, and the XML from the encoding standard (ISO 19118) defines entity tags. All of these may reference the same information content. There is no difficulty in allowing the use of the same name to represent the same information content even though at a deeper level there are significant technical differences in the digital entities being implemented. This “allows” named types defined in the UML model to be used directly under that same name in application schemas.

© ISO 2013 – All rights reserved

There are many conformance options for application schemas that define types for the instantiation of geometric or topological objects. They are differentiated on the basis of the following criteria:

1. Structural complexity,
2. Geometry metrics (geodesy),
3. Topological Dimensionality,
4. Interpolation schemes and
5. Functional completeness.

In defining the geometry metrics, the application schema will be required to specify which underlying surfaces will be supported in determining geometrically based measured. In general, the underlying surfaces (called "Figure of the Earth") are:

1. The 2D plane, or map geometry uses standard Cartesian geometry and Pythagorean "flat" metrics, and has inherent inaccuracies associated to the curvature of the earth dependent on the size of the area covered and the characteristics of the chosen projections (coordinate reference systems or CRS).
2. A 2D sphere uses classical spherical surfaces (a perfectly round earth of constant curvature). These systems use great circles as geodesics and do not adjust for the eccentricity of the earth's surface. Depending on the CRS slightly different spheres may be chosen for a "best fit" for a limited local. While more accurate than planar, as the area of coverage increases accuracy of measurements do degrade but at a slower rate.
3. A 2D ellipsoid uses a surface generated by the rotation of an ellipse on its shorter axis (polar) creating a circular equator and other lines of constant latitude, with orthogonal, elliptical lines of constant longitude. These systems adjust for the eccentricity of the ellipsoid, but do not factor in local gravitational anomalies. Again a variety of "best fit" ellipsoids are used.
4. A 2D geoid or equipotential surface of gravity usually represented as a deviation from a reference ellipsoid.

Three dimensional geometry models will also be supported. There are essentially two approaches:

1. Geocentric 3D, using a 'flat' 3D Euclidean space with a "center" of the Earth defined. These systems are most often used in transformations from one CRS to another CRS based on different surface model.
2. Using one of the surfaces from the 4 classes name above, and adding a measure of the distance in 3D either above or below the reference surface or "Figure of the Earth". The third dimension is measured along a normal line (orthogonal to the surface at its base point).

In defining the topological dimensionality of object types to be implemented, the application schema will be required to specify which of the interpolation types for curves or surfaces they wish to implement.

Curve implementations, for those application schemas including topological 1-dimensional objects, shall always include "linear" or "geodesic" interpolation techniques. Application schema including 1-dimensional objects should always include a mechanism to approximate any curve as a line or geodesic string to allow for transfer of data into simpler schemata when needed.

Surface implementations, for those application schemas including 2-dimensional objects, shall always include a "planar" interpolation technique. The most common such interpolations include triangular irregular networks (TIN) and polyhedral surfaces. Obviously in 2D systems these polygons (with any curves as boundary) are the only surface types. Non-flat surfaces require a 3D

 © ISO 2013 – All rights reserved

coordinate system. Application schema should always include a mechanism to approximate any surface as collections of planar surface patches to allow for transfer of data into simpler schemata when needed. Additional curve and surface interpolation mechanism are optional, but if implemented, they will follow the definition included in this International Standard.

The third criterion (functional completeness) determines the member elements (attributes, association roles and operations) of those types that shall be implemented. The most limited of such schema would define only data types, and may be used in the transfer of data or the passing of operational parameters to service providers.

The first criterion is level of data complexity. Four levels are identified:

1. Geometric primitives
2. Geometric complexes
3. Topological primitives and complexes
4. Topological primitives and complexes with geometric realization

NOTE If single definitions of each component of geometry are required, then geometric complexes are introduced into the schema. Primitives within the same geometric complex share only boundaries. If the schema requires explicit topological information then the geometric complex is expanded to include the structure of a topological complex. The types of object included in a complex are controlled by the dimension of that complex. What is commonly called "chain-node" topology is a 1-dimensional topological complex. What is commonly called "full topology" in a cartographic 2D environment is a 2-dimensional topological complex realized by geometric objects in a 2D coordinate system.

The second criterion is dimensionality. There are four levels for simple spatial geometry:

0. 0-dimensional objects (points and locations in a well-defined coordinate system)
1. 0- and 1-dimensional objects (adding curves of various types)
2. 0-, 1-, and 2-dimensional objects (adding surfaces or areas of various types)
3. 0-, 1-, 2- and 3-dimensional objects (adding solids)

However, 0-dimensional topological or geometric complexes provide no useful information beyond that provided by 0-dimensional geometric primitives, so conformance classes are only defined for complexes of 1-, 2-, and 3-dimensions.

The third criterion is level of functional complexity. There are three levels.

1. Data types only
2. Simple operations
3. Complete operations

Clause 8 (TBD) of this International Standard defines three groups of Boolean operators that may be used to derive topological relations between geometric and topological objects. This International Standard defines four conformance classes for application schemas that implement these operators.

## 2.2 Conformance classes (to be expanded in later drafts)

ED: Each requirements class listed in the UML model will correspond to a conformance class in the Abstract Test suite. Each requirements class dependency will be realized ad a conformance class extension.

To conform to this International Standard, an implementation shall satisfy the requirements of the Abstract test suite (ATS) in Annex A for a specified conformance class and extensions. In general,

© ISO 2013 – All rights reserved

these conformance classes can be given by "descriptor" parameters. These parameters include the following identifiers and their list of possible values

1. Figure of the Earth (coordinate surface representing 0-elevation):
    a. planes,
    b. spheres and planes
    c. ellipsoids (biaxial rotated on the polar axis), spheres and planes; or
    d. geoids, ellipsoids, spheres and planes
2. Mechanism for elevation:
    a. none (2D), or
    b. normal distance from surface (3D).
3. Dimension of non-empty geometry supported:
    a. 0, only points or direct positions
    b. 1, curves and points
    c. 2, surfaces, curves and points, or
    d. 3 solids, surfaces, curves and points
4. Support of topological structures up to dimension
    a. No support
    b. 1,
    c. 2 or
    d. 3.
5. Interpolations
    a. linear, planar, bilinear and trilinear polynomials
    b. Euclidean geometry in local spaces (using the tangent space) which is then mapped to the surface supported using the mapping defined by geodesics (see exponential map in the definitions below (includes geodesics, geodetic circles and other conic sections including spirals defined in polar coordinates).
    c. (to be continued)

This means that there will be one Core class for planar 2D system, and then extensions for each of the parameter values as needed.

## 3 Normative references **(to be updated to most recent versions before publication)**

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

*[1] ISO 19101: 2002, Geographic information — Reference model*
*[2] ISO 19103: 2014, Geographic information — Conceptual schema language (to appear)*
*[3] ISO 19109: 2005, Geographic information — Rules for application schema*
*[4] ISO 19108: 2002,Geographic information — Temporal Schema*
*[5] ISO 19111: 2007, Geographic information — Spatial referencing by coordinates*
*[6] ISO 19111-2: 2009, Geographic information — Spatial referencing by coordinates — Part 2: Extension for parametric values*
*[7] ISO 19125-1:2004 Geographic information — Simple feature access -- Part 1: Common architecture*

 © ISO 2013 – All rights reserved

*[8] ISO 19125-2:2004 Geographic information — Simple feature access -- Part 2: SQL option*
*[9] ISO/IEC 13249-3:2006(E) - Information technology — SQL Multimedia and Application Packages - Part 3: Spatial*
*[10] ISO/IEC 11404: 2007, Information technology — Programming languages, their environments and system software interfaces — Language-independent datatypes*
*[11] OMG Unified Modeling Language™ (OMG-UML), Infrastructure, Version 2.4.1,* http://www.omg.org/spec/UML/2.4/Infrastucture
*[12] OMG Unified Modeling Language™ (OMG-UML), Superstructure, Version 2.4.1,* http://www.omg.org/spec/UML/2.4/Superstructure

## 4 Terms and definitions

ED: Mostly final, default definitions will be those already TMG-approved where available, and applicable, additions may be made.

For the purposes of this document, the following terms and definitions apply. The terms are listed alphabetically in this clause. In Annex B they are organized by their conceptual relationships.

**4.1**
**application**
manipulating and processing of data in support of user requirements [*ISO 19101*]

**4.2**
**application schema**
conceptual schema for data required by one or more **applications [***ISO 19101*]

**4.3**
**bearing (between two points on a FoE)**
direction of initial unit tangent of the geodesic (or other reference curve) from the first point to the second point

Note: When a geodesic tangent is used for bearing, the bearing of the tangent along a geodesic along its path will often vary. If a projection is used to map to a plane, the Line on the plane is not usually a geodesic, in particular if a Mercator projection is used, the line is a rhumb line (loxodrome), a line of constant bearing. While this is handy for navigation, it is not a "shortest line" between the two points on the Earth. For small extends is it sufficiently close not to make significant difference, but the longer the line the bigger the difference.

Usual 2D measure of bearing can be an angle measured from North clockwise, or a unit tangent vector of the geodesic at the first point. If the coordinate system is spatially 3D, the horizontal bearing angle may also need to a vertical altitude angle to be complete. If the geodesic is parameterized by arc length, then the "derivative" of the geodesic is a unit vector. If another parameterization is used, then the derivative may be normalized ( $\vec{T} / \|\vec{T}\|$ ). This is useful, since parameterization by arc length can be computationally difficult. Any parameterization is allowed as long at the parameter does not have an inflection point (zero derivative) with respect to arc length.

The bearing is equal to the relative bearing from north. In coordinate spaces, angles are measured from the first axis to the second. Hence in classical Cartesian planes, where the x-axis is horizontal, angles are positive counter clockwise (the bearing of the y-axis is +90°). In more classical latitude, longitude system, the bearing is positive clockwise, the direction from the latitude axis (north) to the longitude axis (east).

The type of curve is often determined by tradition. Mercator projections use rhumb lines (loxodromes) which are straight lines in this projection. Some more modern geodetic system use geodesics. Systems will have to be precise in the use of the term, signifying the curve type, and absolute or relative bearings.

**4.4**
**boundary**
**set** that represents the limit of an entity

© ISO 2013 – All rights reserved

NOTE **Boundary** is most commonly used in the context of geometry, where the set is a collection of points or a collection of objects that represent those points. In other arenas, the term is used metaphorically to describe the transition between an entity and the rest of its domain of discourse.

**4.5**
**buffer**
**geometric object** that contains all **DirectPositions** whose distance from a specified **geometric object** is less than or equal to a given distance

**4.6**
**circular sequence**
**sequence** which has no logical beginning and is therefore equivalent to any circular shift of itself; hence the last item in the sequence is considered to precede the first item in the sequence

**4.7**
**class**
description of a **set** of object**s** that share the same attributes, operations, methods, relationships, and semantics [*ISO 19103*]

NOTE A class may use a **set** of **interfaces** to specify collections of **operations** it provides to its environment. The term was first used in this way in the general theory of object oriented programming, and later adopted for use in this same sense in UML.

Derived from *OMG-UML*

**4.8**
**classifier**
description of a **set** of entities that share the structural and behavioral semantics

NOTE Classifiers are usually represented graphically as classes are, but often have specific stereotypes or properties that differentiate behavior and instantiation mechanisms from a more common class model.

Derived from *OMG-UML*

**4.9**
**closure**
union of the **interior** and **boundary** of a **topological** or **geometric object**

**4.10**
**coboundary**
**set** of **topological primitives** of higher topological dimension associated with a particular **topological object**, such that this **topological object** is in each of their **boundaries**

NOTE If a node is on the boundary of an edge, that edge is on the coboundary of that node. Any orientation parameter associated to one of these relations would also be associated to the other. So that if the node is the end node of the edge (defined as the end of the positive directed edge), then the positive orientation of the node (defined as the positive directed node) would have the edge on its coboundary, see Figure 35.

**4.11**
**composite curve**
sequence of **curves** such that each curve (except the first) starts at the end point of the previous curve in the sequence

NOTE A composite curve, as a set of **DirectPositions**, has all the properties of a curve.

 © ISO 2013 – All rights reserved

**4.12**
**composite solid**
connected **set** of **solids** adjoining one another along shared **boundary surfaces**

NOTE A composite solid, as a set of **DirectPositions**, has all the properties of a solid.

**4.13**
**composite surface**
connected **set** of **surfaces** adjoining one another along shared **boundary curves**

NOTE A composite surface, as a set of **DirectPositions**, has all the properties of a surface.

**4.14 (?needed?)**
**computational geometry**
manipulation of and calculations with geometric representations for the implementation of geometric **operations**

EXAMPLE Computational geometry operations include testing for geometric inclusion or intersection, the calculation of **convex hulls** or **buffer** zones, or the finding of shortest distances between **geometric objects**.

**4.15**
**computational topology(?needed?)**
topological concepts, structures and algebra that aid, enhance or define **operations** on **topological objects** usually performed in **computational geometry**

**4.16**
**connected**
property of a **geometric object** implying that any two **DirectPositions** on the object can be placed on a **curve** that remains totally within the object

NOTE A topological object is connected if and only if all its **geometric realizations** are connected. This is not included as a definition because it follows from a theorem of topology.

**4.17**
**connected node**
**node** that starts or ends one or more **edges**

**4.18**
**control point**
position used in the construction of a geometry that partially controls its shape but does not necessarily lie on the geometry

Note A center of an arc is a control point; poles in b-spline curves are control points

**4.19**
**convex hull**
smallest **convex set** containing a given **geometric object [**Dictionary of Computing [7]]

NOTE "Smallest" is the set theoretic smallest, not an indication of a measurement. The definition can be rewritten as "the intersection of all convex sets that contain the geometric object".

© ISO 2013 – All rights reserved

**4.20**
**convex set**
**geometric set** in which any direct position on the geodesic segment joining any two direct position in the geometric set is also contained in the **geometric set [**Dictionary of Computing **Error! Reference source not found.**]

NOTE Convex sets are "simply connected", meaning that they have no interior holes, and can normally be considered topologically isomorphic to a Euclidean ball of the appropriate dimension. So the surface of a sphere can be considered to be geodesically convex.

The term is usually defined by "line segments joining any two positions" but this is more applicable to flat Euclidean spaces.

**4.21**
**coordinate**
one of a **sequence** of N-numbers designating the position of a **point** in N-dimensional space [ISO 19111]

NOTE In a **coordinate reference system**, the numbers must be qualified by units.

**4.22**
**coordinate dimension**
number of measurements or axes needed to describe a position in a **coordinate system**

**4.23**
**coordinate reference system**
**coordinate system** that is related to the real world by a datum [*ISO 19111*]

**4.24**
**coordinate system**
set of mathematical rules for specifying how **coordinates** are to be assigned to **points [***ISO 19111*]

**4.25**
**coplanar**
lying in the same geodesic plane

**4.26**
**curvature vector (of a curve in $\mathbb{E}^3$)**
second derivative of a curve parameterized by arc length, at a point

Notation: If $\boldsymbol{c}(s)=(x(s),y(s),z(s))$ $\boldsymbol{c}(s)=(x(s),y(s),z(s))$ $\boldsymbol{c}(s)=(x(s),y(s),z(s))$ is a curve in a 3D Cartesian space ($\mathbb{E}^3$), and $s$ is the arc length along $c$, then the tangent is $\boldsymbol{c}(s)=(x(s),y(s),z(s))$ $\dot{\boldsymbol{c}}(s)=(\dot{x}(s),\dot{y}(s),\dot{z}(s))$, i.e. the derivative of the coordinate values of $c$ with respect to $s$. The tangent can be approximated by a normalized directed line between any two points on the curve near the point.

The curvature vector is $\ddot{\boldsymbol{c}}(s)=(\ddot{x}(s),\ddot{y}(s),\ddot{z}(s))$. The curvature vector can be approximated by the radius of a circle through any 3 nearby points on the curve (pointed from the curve to towards the center of the circle)

**4.27**
**curve**
1-dimensional **geometric primitive**, representing the continuous image of a line


© ISO 2013 – All rights reserved

NOTE The **boundary** of a **curve** is the **set** of **points** at either end of the **curve**. If the curve is a cycle, the two ends are identical, and the curve (if topologically closed) is considered to not have a boundary. The first **point** is called the **start point**, and the last is the **end point**. Connectivity of the curve is guaranteed by the "continuous image of a line" clause. A topological theorem states that a continuous image of a connected set is connected.

**4.28**
**cycle**
**(geometry)**
**spatial object** with an empty **boundary**

NOTE Cycles are used to describe boundary components (see **shell**, **ring**). A cycle has no boundary because it closes on itself, but it is bounded (i.e., it does not have infinite extent). A circle or a sphere, for example, has no boundary, but is bounded.

**4.29**
**data point**
position used in the construction of a geometry that lie on the geometry

Note The vertices in a line string are data points, the points uses to construct a polynomial spline are data points. Data points can be used as control points, but are often derived after the geometry is constructed.

**4.30**
**direct position**
position that can be described by a single set of **coordinates** within a **coordinate reference system**

**4.31**
**directed edge**
**directed topological object** that represents an association between an **edge** and one of its orientations

NOTE A directed edge that is in agreement with the orientation of the edge has a + orientation, otherwise, it has the opposite (–) orientation. Directed edge is used in **topology** to distinguish the right side (–) from the left side (+) of the same edge and the **start node** (–) and **end node** (+) of the same edge and in **computational topology** to represent these concepts.

**4.32**
**directed face**
**directed topological object** that represents an association between a **face** and one of its orientations

NOTE The orientation of the **directed edges** that compose the exterior **boundary** of a directed face will appear positive from the direction of this vector; the orientation of a directed face that bounds a **topological solid** will point away from the **topological solid**. Adjacent solids would use different orientations for their shared boundary, consistent with the same sort of association between adjacent faces and their shared edges. Directed faces are used in the **coboundary** relation to maintain the spatial association between **face** and **edge**.

**4.33**
**directed node**
**directed topological object** that represents an association between a **node** and one of its orientations

NOTE Directed nodes are used in the **coboundary** relation to maintain the spatial association between **edge** and **node**. The orientation of a node is with respect to an edge, "+" for end node, "–" for start node. This is consistent with the vector notion of "result = end - start".

**4.34**
**directed solid**
**directed topological object** that represents an association between a **topological solid** and one of its orientations

© ISO 2013 – All rights reserved

NOTE Directed solids are used in the **coboundary** relation to maintain the spatial association between **face** and **topological solid**. The orientation of a solid is with respect to a face, "+" if the upNormal is outward, "–" if inward. This is consistent with the concept of "up = outward" for a surface bounding a solid.

**4.35**
**directed topological object**
**topological object** that represents a logical association between a **topological primitive** and one of its orientations

**4.36**
**distance (between points on a FoE)**
minimal arc length of a curve that joins the two points

Note The usual distance function in a coordinate space assumes an underlying plane and is often referred to as a Euclidean distance. If the underlying FoE is not a plane, then distance is defined by this minimum length of all curves.

**4.37**
**distance (between any two geometric objects on the same FoE)**
largest length smaller than all lengths of curves from a point in one geometric object to a point in the other geometric object

Note In mathematical terms, this is the "greatest lower bound" of the length of the curves.

**4.38**
**domain**
well-defined **set [**ISO/TS 19103]

NOTE Domains are used to define the domain and range of operators and **functions**.

**4.39**
**edge**
1-dimensional **topological primitive**

NOTE The **geometric realization** of an edge is a **curve**. The **boundary** of an edge is the **set** of one or two **nodes** associated to the edge within a **topological complex**.

**4.40**
**edge-node graph**
**graph** embedded within a **topological complex** composed of all of the **edges** and **connected nodes** within that **complex**

NOTE The **edge-node graph** is a subcomplex of the complex within which it is embedded.

**4.41**
**ellipsoid FoE**
figure of the earth represented by an ellipsoid of revolution, that is a surface of rotation around the polar axis so that the equatorial radii are all equal, embedded in 3D Euclidean space with center (half way between the foci) located at the nominal center of gravity of the earth, latitude is measured by the angle of intersection of the northward tangent and the polar axis of revolution, longitude is measured as the angular offset from the prime meridian as is done in the spherical case

**4.42**
**empty set**
set without any elements, symbolized by ∅


© ISO 2013 – All rights reserved

Math note: Sets are equal if they contain exactly the same elements. Since any two empty sets would share exactly the same contained elements (by definition none), they would be, by definition, be equal.

The empty set (∅) can be considered a geometry entity, because all of the elements it contain are direct positions. This is a vacuous statement since the set ∅ contains no elements, and therefore the "for all" statement has nothing to test and is thus true in each of its non-existent cases. There are a lot of true vacuous statements in proofs about ∅.

Note This confuses some programmers since many systems use type-safe sets, in which the class of the entities determines a class for the container set. The math does not care about "class" and only sees sets; so that an empty set of aardvarks and an empty set of zebras in mathematics are (is?) the same set.

The other confusion is that ∅ is not the database Null introduced by Codd and used in relational and other query languages in 3-valued logic. Null means unknown and many statements involving Null are undesirable. The empty set is not "lack of knowledge" but certainty in the nonexistence of elements in the set. Most statements beginning "for all elements in ∅" are true, but vacuous. Most statement beginning "there exist an element in ∅" are always categorically false. It is almost impossible to construct an undecidable statement about ∅. Null and ∅ are not related.

**4.43**
**end node**
**node** in the **boundary** of an **edge** that corresponds to the **end point** of that **edge** as a **curve** in any valid **geometric realization** of a **topological complex** in which the **edge** is used

**4.44**
**end point**
last point of a **curve**

**4.45**
**error budget**
statement of or methodology for describing the nature and magnitude of the errors which affect the results of a calculation

NOTE In the most usual case, **error budgets** in this standard describe metric calculations using representational geometry objects to estimate real-world metrics, such as distance and area.

**4.46**
**exponential map [differential geometry]**
function that maps tangent vectors at a point to end point of geodesic beginning at that point with a exit bearing equal to that of the vector and a length equal to that of the vector [see first geodetic problem]

**4.47**
**exterior**
difference between the universe and the **closure**

NOTE The concept of exterior is applicable to both **topological** and **geometric complexes**.

**4.48**
**face**
2-dimensional **topological primitive**

NOTE The **geometric realization** of a face is a **surface**. The **boundary** of a face is the **set** of **directed edges** within the same **topological complex** that are associated to the face via the boundary relations. These can be organized as **rings**.

**4.49**
**feature**
abstraction of real world phenomena [ISO 19101]

© ISO 2013 – All rights reserved

NOTE A feature may occur as a type or an **instance**. Feature type or feature instance should be used when only one is meant.

**4.50**
**feature attribute**
characteristic of a **feature [**ISO 19101]

NOTE A feature attribute has a name, a data type, and a value **domain** associated to it. A feature attribute for a feature **instance** also has an attribute value taken from the value domain.

**4.51**
**first geodetic problem**
problem that given a point (in terms of its coordinates) on a Figure of the Earth and the direction (azimuth or bearing) and distance from that point to a second point, determine (the coordinates of) that second point

Note This "problem" defines a mapping from the vector space at a point (each vector given by a direction and a length) to points of the Figure of Earth that satisfy the problem for that direction and distance. For example if we fix the distance "r" and take all directions, the resultant geometry is the circle centered at the original point of radius "r."

This standard will make heavy use of this mapping; see exponential map

**4.52**
**FoE (Figure of the Earth)**
Surface in some Euclidean space that represents an approximation to the surface of the earth possibly restricted to a small area but often covering the entire globe.

**4.53**
**footprint**
2D horizontal extent of an object

**4.54**
**function**
rule that associates each element from a **domain** (source, or domain of the function) to a unique element in another domain (target, co-domain, or range)

**4.55**
**geodesic, gedesic line**
**curve** on a surface with a zero length tangential curvature vector

Note A geodesic's curvature vector is perpendicular the surface thus has the minimum curvature of any curve restricted to the surface.

**4.56**
**geodesic plane, geodesic surface**
surface in which any two points are joined by a **geodesic** contained completely within the surface

Note If the figure of the earth is 2D, then it is, by definition, a geodesic surface.

**4.57**
**geodetic circle**
set of points an equal distance from a given point (on the Figure of the Earth)

Note The geodetic circles centered on a pole (either one) are the lines of constant latitude. Circles in a tangent space centered on the origin (corresponding to the point of tangency) map to geodetic circles on the Figure of the Earth centered on the point of tangency.


© ISO 2013 – All rights reserved

**4.58**
**geoid FoE**
equipotential surface of gravity i.e. a surface with the acceleration of gravity is constant (standard gravity, or about 9.80665 $m/s^2$) usual representation is as an offset numeric surface from a reference ellipsoid

NOTE Latitude and longitude are measured using the reference ellipsoid (for example GPS will be ellipsoid coordinates with the offset from the ellipsoid and not the geoid).

**4.59**
**geographic information**
information concerning phenomena implicitly or explicitly associated with a location relative to the Earth [ISO 19101]

**4.60**
**geometric aggregate**
collection of **geometric objects** that has no internal structure

NOTE No assumptions about the spatial relationships between the elements can be made.

**4.61**
**geometric boundary**
**boundary** represented by a **set** of **geometric primitives** that limits the extent of a **geometric object**

**4.62**
**geometric centroid (of a geometric object)**
**barycenter**
calculated "center of mass" of the geometric objects maximal dimensional components based on an assumption of uniform density, average location of all points in the object

Note The restricted use of the points in the maximal dimensions components is a result that the mathematical integral over an n-dimensional extent, when applied to any lower dimension is zero.

**4.63**
**geometric complex**
**set** of disjoint **geometric primitives** where the **boundary** of each **geometric primitive** can be represented as the union of other **geometric primitives** of smaller dimension within the same **set**

NOTE The **geometric primitives** in the set are disjoint in the sense that no direct position is **interior** to more than one **geometric primitive**. The set is closed under **boundary operations**, meaning that for each element in the **geometric complex**, there is a collection (also a **geometric complex**) of **geometric primitives** that represents the boundary of that element. Recall that the **boundary** of a point (the only 0D primitive object type in geometry) is empty. Thus, if the largest dimension geometric primitive is a solid (3D), the composition of the boundary operator in this definition terminates after at most three steps. It is also the case that the boundary of any object is a **cycle**.

**4.64**
**geometric dimension**
largest number *n* such that each **direct position** in a **geometric set** can be associated with a subset that has the **direct position** in its **interior** and is isomorphic to $R^n$, Euclidean *n*-space

NOTE Curves, because they are continuous images of a portion of the real line, have geometric dimension 1. Surfaces cannot be mapped to $R^2$ in their entirety, but around each point position, a small neighbourhood can be found that resembles (under continuous functions) the interior of the unit circle in $R^2$, and are therefore 2-dimensional. In this International Standard, most surfaces (instances of Surface) are mapped to portions of $R^2$ by their defining interpolation mechanisms.

© ISO 2013 – All rights reserved

**4.65**
**geometric object**
**spatial object** representing a **geometric set**

NOTE A **geometric object** consists of a **geometric primitive**, a collection of **geometric primitives**, or a geometric complex treated as a single entity. A geometric object may be the spatial representation of an **object** such as a **feature** or a significant part of a **feature**.

**4.66**
**geometric primitive**
**geometric object** representing a single, **connected**, homogeneous element of space

NOTE Geometric primitives are non-decomposed **objects** that present information about geometric configuration. They include **points**, **curves**, **surfaces**, and **solids**. Many Geometric objects behave like primitives (supporting the same interfaces defined for geometric primitives) but are actually composites composed of some number of other primitives. General collections may be aggregates and incapable of acting like a primitive (such as the lines of a complex network, which is not connected and thus incapable of being traceable as a single line).

**4.67**
**geometric realization**
**geometric complex** whose **geometric primitives** are in a 1-to-1 correspondence to the **topological primitives** of a **topological complex**, such that the **boundary** relations in the two complexes agree

NOTE In such a realization the topological primitives are considered to represent the **interiors** of the corresponding geometric primitives. Composites are closed.

**4.68**
**geometric set**
**set** of **DirectPositions**

NOTE This set in most cases is infinite, except where the set consists of a list of point locations. Curves, surfaces and volumes being continuous are infinite sets of points. Some systems will define 'degenerate' curves, which are actually points.

**4.69**
**graph**
**set** of **nodes**, some of which are joined by **edges**

NOTE In geographic information systems, a graph can have more than one **edge** joining two **nodes**, and can have an **edge** that has the same **node** at both ends.

**4.70**
**homomorphism**
relationship between two **domains** (such as two complexes) such that there is a structure-preserving **function** from one to the other

NOTE **Homomorphisms** are distinct from **isomorphisms** in that no inverse function is required. In an isomorphism, there are essentially two **homomorphisms** that are functional inverses of one another. Continuous functions are topological homomorphisms because they preserve "topological characteristics". The mapping of topological complexes to their geometric realizations preserves the concept of boundary and is therefore a homomorphism.

**4.71**
**inner product**
bilinear, symmetric function from pairs of vectors $\langle \vec{v}_1, \vec{v}_2 \rangle \rightarrow \mathbb{R}$ to a real number such that $\langle \vec{v}, \vec{v} \rangle = \|\vec{v}\|$ and $\langle \vec{v}_1, \vec{v}_2 \rangle = \|\vec{v}_1\| \|\vec{v}_2\| \cos\theta$ where $\theta$ is the angle between $\vec{v}_1$ and $\vec{v}_2$.


© ISO 2013 – All rights reserved

NOTE **Inner products** in differential geometry the inner product is used on the differentials that make up the local tangent spaces. In this International Standard, this will usually be vectors tangent to a Figure of the Earth surface in a geocentric $\mathbb{E}^3$ Euclidean/Cartesian space.

**4.72**
**instance**
**object** that realizes a **classifier**

**4.73**
**interior**
**set** of all **direct positions** that are on a **geometric object** but which are not on its **boundary**

NOTE The **interior** of a **topological object** is the homomorphic image of the interior of any of its **geometric realizations**. This is not included as a definition because it follows from a theorem of topology.

**4.74**
**isolated node**
**node** not related to any **edge**

**4.75**
**isometry**
mapping between metric spaces that preserves the metric

Note $f: X \to Y \ni \forall x, y \in X,\ distance(x, y) = distance(f(x), f(y))$

**4.76**
**isomorphism**
relationship between two **domains** (such as two complexes) such that there are 1-to-1, structure-preserving **functions** from each **domain** onto the other, and the composition of the two **functions**, in either order, is the corresponding identity function

NOTE A **geometric complex** is isomorphic to a **topological complex** if their elements are in a 1-to-1, dimension- and **boundary**-preserving correspondence to one another.

**4.77**
**loxodrome (rhumb line)**
curve with constant **bearing**, crossing each longitude and each latitude line at the same angles

**4.78**
**maximum (max)**
**least upper bound (lub) of a set A [Mathematics]**
in an ordered domain (≤), the smallest element larger than or equal to all elements of A

Math: $\left[\forall \mathrm{a} \in \mathrm{A},\ \max(A) \geq a\right] \wedge \left[\left[\exists b \ni \left[(a \in A) \Rightarrow (b \geq a)\right]\right] \Rightarrow \max(\mathrm{A}) \leq b\right]$

Note Any number is an upper bound of ∅ (empty set) as a set of numbers, because any given number is greater than any number in ∅ (an admitted vacuous statement since there is no number in ∅, but true none the less). This means that the max(∅) must be smaller than any number. The symbol for this is "-∞".

© ISO 2013 – All rights reserved

**4.79**
**minimum (min)**
**greatest lower bound (glb) of a set A [Mathematics]**
in an ordered domain (≤), the largest element smaller than or equal to all elements of A

Math: $\left[\forall \mathrm{a} \in \mathrm{A},\ \min(A) \le a\right] \wedge \left[\left[\exists b \ni \left[(a \in A) \Rightarrow (b \le a)\right]\right] \Rightarrow \min(A) \ge b\right]$

Note Any number is a lower bound of ∅ considered as a set of numbers, because any given number is less than any number in ∅ (an admitted vacuous statement since there is no number in ∅, but true none the less). This means that the min(∅) must be greater than any number. The symbol for this is "+∞".

**4.80**
**neighbourhood**
**geometric set** containing a specified **direct position** in its **interior**, and containing all DirectPositions within a specified distance of the specified direct position

**4.81**
**node**
0-dimensional **topological primitive**

NOTE The **boundary** of a node is the empty **set**.

**4.82**
**normal curvature vector (of a curve on a surface at a point in $\mathbb{E}^3$)**
projection of the curvature vector of the curve perpendicular to the tangent plane to the surface at the point (and thus parallel to the normal vector of the surface)

Note The normal curvature of a curve is dependent only on the surface (which is a constraint of the curve)

**4.83**
**normal (to surface at a point in $\mathbb{E}^3$)**
vector perpendicular (orthogonal) to the surface at the point

Note The normal will be parallel to the "upward" normal for surfaces in $\mathbb{E}^3$ define in the surface interface in clause 6.3.12.

**4.84**
**object**
entity with a well-defined **boundary** and identity that encapsulates state and behavior

[*OMG-UML*]

NOTE This term was first used in this way in the general theory of object oriented programming, and later adopted for use in this same sense in UML. An object is an instance of a **class**. Attributes and relationships represent state. Operations, methods, and state machines represent behavior.

**4.85**
**planar topological complex**
**topological complex** that has a **geometric realization** that can be embedded in Euclidean 2 space

**4.86**
**point**
0-dimensional **geometric primitive**, representing a position

 © ISO 2013 – All rights reserved

NOTE The **boundary** of a point is the empty set.

**4.87**
**record**
finite, named collection of related items (**objects** or values)

NOTE Logically, a record is a set of pairs <name, item>.

**4.88**
**relative bearing**
angle measured clockwise from the nominal direction of travel (forward or **tangent**) to line of sight of an observation

Note A positive measure is to the right of the direction of motion, and a negative one is to the left. Normally, the range of a relative bearing is $[-180°,+180°]$ but any representation using an added or subtracted multiple of $360°$ is valid.

[from nautical navigation]

**4.89**
**rhumb line (loxodrome)**
**curve** which crosses each meridians of longitude at the same **bearing**

**4.90**
**ring**
**simple curve** which is also a **cycle**

NOTE Rings are used to describe **boundary** components of surfaces in 2D and 3D **coordinate systems**. A common phrase used to describe a cycle is "simple closed curve."

**4.91**
**row-major form**
storage mechanism for multidimensional array in linear memory, organized such that each row is stored in consecutive locations and such that the complete rows are the stored one after the other

Note: If the indexes are (i, j) with the number of rows "r" and columns "c", then the mapping between the multidimensional locations to the linear storage locations is given by:

$$i, j \in \mathbb{Z} \ni 0 \le i < r; 0 \le j < c$$
$$(i, j) \to j + c \cdot i$$

Example: The matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ in row major form is stored as $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}$.

For higher dimensions, the same pattern is applied recursively:

$$i, j, k \in \mathbb{Z} \ni 0 \le i < r; 0 \le j < c; 0 < k < f$$
$$(i, j, k) \to k + f(j + c \cdot i)$$

© ISO 2013 – All rights reserved

**4.92**
**second geodetic problem**
problem that given two points, determine the initial azimuth (bearing) and length of the geodesic that connects them

**4.93**
**sequence**
finite, ordered collection of related items (**objects** or values) that may be repeated

NOTE Logically, a sequence is a set of pairs <item, offset>. LISP syntax, which delimits sequences with parentheses and separates elements in the sequence with commas, is used in this International Standard.

**4.94**
**set**
well-defined collection of distinct entities, considered as an entity in its own right

unordered (potentially infinite) collection of related items (**objects** or values) with no repetition

Note The first definition is useful in mathematical set theory; the second is equivalent in a programming environment. If the set as an object is infinite, it will be defined by a Boolean statement that specifies the domain of the values and the test that if TRUE indicates the value is in the set, and if FALSE the value is not.

**4.95**
**shell**
**simple surface** which is a cycle

NOTE Shells are used to describe **boundary** components of solids in 3D **coordinate systems**.

**4.96**
**simple**
property of a **geometric object** that its **interior** is isotropic (all points have **isomorphic** neighborhoods) and its boundary is also simple.

Note The interior is everywhere locally **isomorphic** to an open disc in a Euclidean coordinate space of the appropriate dimension $D^n = \{P \mid \|P\| < 1.0\}$ . The simple for the boundary would usually have to use a dimension one smaller.

**4.97**
**solid**
3-dimensional **geometric primitive**, representing the continuous image of a region of Euclidean 3 space

NOTE A **solid** is realizable locally as a three parameter **set** of **DirectPositions**. The **boundary** of a **solid** is the set of oriented, closed **surfaces** that comprise the limits of the **solid**.

**4.98**
**spatial object**
**object** used for representing a spatial characteristic of a feature

**4.99**
**spatial operator**
function or procedure that has at least one spatial parameter in its **domain** or range

NOTE Any UML operation on a spatial object would be classified as a spatial operator as are the query operators in Clause 8 of this International Standard.


© ISO 2013 – All rights reserved

**4.100**
**sphere FoE**
figure of the earth represented by a sphere, that is the loci of points in 3D Euclidean space at a fixed distance from the nominal center of gravity of the earth; associated to a latitude-longitude coordinate system where the angle are all measured as the central angles of the radial line from the center to the point on the earth, latitude is the angular offset from the equatorial plane, longitude measured as the angular offset from the prime meridian.

**4.101**
**start node**
**node** in the **boundary** of an **edge** that corresponds to the **start point** of that **edge** as a **curve** in a valid **geometric realization** of the **topological complex** in which the **edge** is used

**4.102**
**start point**
first **point** of a **curve**

**4.103**
**strong substitutability**
ability for any **instance** of a **class** that is a descendant under inheritance or realization of another **class**, type or **interface** to be used in lieu of an **instance** of its ancestor in any context

NOTE The weaker forms of substitutability make various restrictions on the context of the implied substitution.

**4.104**
**subcomplex**
**complex** all of whose elements are also in a larger complex

NOTE Since the definitions of **geometric complex** and **topological complex** require only that they be closed under **boundary operations**, the **set** of any primitives of a particular dimension and below is always a subcomplex of the original, larger complex. Thus, any full **planar topological complex** contains an **edge-node graph** as a subcomplex.

**4.105**
**surface**
2-dimensional **geometric primitive**, locally representing a continuous image of a region of a plane

NOTE The **boundary** of a **surface** is the set of oriented, closed **curves** that delineate the limits of the **surface**. **Surfaces** that are isomorphic to a sphere, or to an n-torus (a topological sphere with n "handles") have no boundary. Such surfaces are called **cycles**.

**4.106**
**surface patch**
2-dimensional, **connected geometric object** used to represent a continuous portion of a **surface** using homogeneous interpolation and definition methods

**4.107**
**tangent (of a curve at a point)**
direction indicating the instantaneous direction of a curve at a point

Note The tangent is usually calculated by differentiation of a functional representation of a curve but it may be approximated by a secant (double intersection) line from the point passing through another nearby point on the curve. The closer the second point is to the first, the better the approximation of the tangent.

© ISO 2013 – All rights reserved

**4.108**
**tangential curvature vector,**
**geodesic curvature vector (of a curve on a surface at a point in $\mathbb{E}^3$)**
projection of the **curvature vector** of the curve onto the tangent plane to the surface at the point

**4.109**
**tangent space (of a surface at a point)**
collection of tangent vectors for curves passing through the point

**4.110**
**tanget vector of a curve in $\mathbb{E}^3$**
first derivative of a curve parameterized by arc length

Notation: If $\boldsymbol{c}(\boldsymbol{s}) = (\boldsymbol{x}(\boldsymbol{s}), \boldsymbol{y}(\boldsymbol{s}), \boldsymbol{z}(\boldsymbol{s}))$ is a curve in a 3D Cartesian space ($\mathbb{E}^3$), and $s$ is chosen to be the arc length along $c$, then the tangent is $\tau(s) = \dot{\boldsymbol{c}}(\boldsymbol{s}) = (\dot{\boldsymbol{x}}(\boldsymbol{s}), \dot{\boldsymbol{y}}(\boldsymbol{s}), \dot{\boldsymbol{z}}(\boldsymbol{s}))$, i.e. the derivative of the coordinate values of $c$ with respect to $s$. The curvature vector is $\kappa(s) = \ddot{\boldsymbol{c}}(\boldsymbol{s}) = (\ddot{\boldsymbol{x}}(\boldsymbol{s}), \ddot{\boldsymbol{y}}(\boldsymbol{s}), \ddot{\boldsymbol{z}}(\boldsymbol{s}))$.

**4.111**
**topological boundary**
**boundary** represented by a **set** of oriented **topological primitives** of smaller topological dimension that limits the extent of a **topological object,** or by the **geometric realization** of such a set

NOTE The **boundary** of a **topological complex** corresponds to the boundary of the **geometric realization** of the topological complex.

**4.112**
**topological complex**
collection of **topological primitives** that is closed under the **boundary** operations

NOTE Closed under the boundary operations means that if a **topological primitive** is in the **topological complex**, then its **boundary** objects are also in the **topological complex**.

**4.113**
**topological dimension**
minimum number of free variables needed to distinguish nearby **DirectPositions** within a **geometric object** from one another

NOTE The free variables mentioned above can usually be thought of as a local coordinate system. In a 3D coordinate space, a plane can be written as $P(u, v) = A + u X + v Y$, where $u$ and $v$ are real numbers and $A$ is any point on the plane, and $X$ and $Y$ are two vectors tangent to the plane. Since the locations on the plane can be distinguished by $u$ and $v$ (here universally), the plane is 2D and $(u, v)$ is a coordinate system for the points on the plane. On generic surfaces, this cannot, in general, be done universally. If we take a plane tangent to the surface, and project points on the surface onto this plane, we will normally get a local isomorphism for small neighbourhoods of the point of tangency. This "local coordinate" system for the underlying surface is sufficient to establish the surface as a 2D topological object.

Since this International Standard deals only with spatial coordinates, any 3D object can rely on coordinates to establish its topological dimension. In a 4D model (spatio-temporal), tangent spaces also play an important role in establishing topological dimension for objects up to 3D.

**4.114**
**topological expression**
collection of oriented **topological primitives** which is operated upon like a multivariate polynomial

 © ISO 2013 – All rights reserved

NOTE Topological expressions are used for many calculations in **computational topology**.

**4.115**
**topological object**
**spatial object** representing spatial characteristics that are invariant under continuous transformations

NOTE A **topological object** is a **topological primitive**, a collection of topological primitives, or a **topological complex**.

**4.116**
**topological primitive**
**topological object** that represents a single, non-decomposable element

NOTE A topological primitive corresponds to the interior of a geometric primitive of the same dimension in a geometric realization.

**4.117**
**topological solid**
3-dimensional **topological primitive**

NOTE The **boundary** of a topological solid consists of a set of **directed faces**.

**4.118**
**universal face**
unbounded **face** in a 2-dimensional complex

NOTE The **universal face** is normally not part of any feature, and is used to represent the unbounded portion of the data set. Its interior boundary (it has no exterior boundary) would normally be considered the exterior boundary of the map represented by the data set. This International Standard does not special case the **universal face**, but application schemas may find it convenient to do so.

**4.119**
**universal solid**
unbounded **topological solid** in a 3-dimensional complex

NOTE The **universal solid** is the 3-dimensional counterpart of the universal face, and is also normally not part of any feature.

**4.120**
**vector geometry**
representation of **geometry** through the use of constructive **geometric primitives**

# 5 Symbols, notation and abbreviated terms

## 5.1 Presentation and notation

### 5.1.1 Unified Modeling Language (UML) concepts

In this International Standard, conceptual schemas are presented in the Unified Modeling Language (UML). In general, UML 2 is used and most of the classifiers in ISO 19107: 2003 which are stereotyped «type» are stereotyped «interface» in this international standard.

The fact that an element of geometry or topology has been modeled in one way or another in this International Standard should not be considered a restriction on implementations. Attributes may be implemented either directly as data or as a pair of “accessor” and “mutator” operations for getting and setting values. Most diagrams in this document are “context diagrams” which center about a single class and display its attributes, operations, and important relationships. Other diagrams are overviews of class relationships. UML does not require all relationships to be displayed in all

© ISO 2013 – All rights reserved

diagrams, and some of the more trivial ones have been left out of many diagrams to keep them simple. For example, GeometryObject has an obvious relationship to Set<GeometryObject>, but this is not explicitly shown in the context diagram for GeometryObject.

### 5.1.2 Specialization

The most common relation between classes is specialization by subclassing or realization. The criteria for specialization should always be based on the internal structure and function. This means that an instance of a subclass can always be used as if it were an instance of any of its superclasses; this is called "substitutability." Specialization should never reflect the external use of the new class, but should reflect only internal extensions and additional functionality. Where possible classes defined in this model have been given aliases that reflect the classes in earlier standards that they replace.

The use of specialization to reflect usage is a common logical error. The use of "boundary types" in the ISO 19107: 2003 is an example of this error that has been corrected in this edition of the standard.

### 5.1.3 UML usage

This standard uses UML 2 as it is defined by the OMG. Some standard extensions are used to reflect common geographic usage.

Object-oriented operator notation (such as would be used in C++) places the first parameter before the operation as in a method declaration as follows:

```
return-type type-1::operation(type-2, type-3 … )
```

Such methods are restricted in name space, in the sense that they are available only if the "object-type-1" name space is available. In addition, during invocation, the identification of the implicit parameter of type "type-1" is known. In OCL, this object is identified as "self". In C++, this object is identified as "this". In non-object languages or for free functions in an object language, the functional notation for an operation does not distinguish the first parameter in any manner and is written:

```
operation(name-1:type-1, name-2:type-2, name-3:type-3 …):return-
     type, …
```

These notations are equivalent (except for emphasis) and both may be used in profiles of this International Standard.

These operation definitions are called "operation signatures" or "protocols". This distinguishes the operation from the invocation mechanism. In UML, the formal notation defines protocols, and the operations associated to them are defined only informally in the associated documentation, which can include object constrain language (OCL) constraints. The identity of a protocol depends both on its name, and the types of all of its parameters. The mapping from invocation to actual function execution is a system facility, and can usually be treated as transparent to the user/programmer.

In the view that an attribute can be considered a type of operation (mutator and accessor pairs), this term can be extended to include attribute "signatures". The definition of a signature includes the operation name; the parameter names and types; and the return type. Methods or attributes can be overridden by providing a new method whose signature is the same as the original except that some of the types have been replaced, usually by subtypes of the originals. The reuse of signatures is called


© ISO 2013 – All rights reserved

"polymorphism". Polymorphism arising from class inheritance is called "structural". Polymorphism arising from semantic similarity is called "natural" or "generic". For example, in the geometry and topology classes, the common protocol for "boundary" is a natural polymorphism in that it arises from an operational constraint based on the definition of topology. It is not a structural polymorphism, since the two packages do not share a common superclass ancestor. Assuming that the class inheritance hierarchy is based on semantics of the objects, then structural polymorphism is natural. Polymorphism that does not depend on semantic similarity is "ad-hoc". For example, the use of "+" in numbers to denote addition and in character string classes to denote concatenation is ad-hoc polymorphism. Ad-hoc polymorphism is semantically confusing, and is therefore not used in this International Standard, and should be avoided in profiles of this International Standard.

Most operations are defined in a functional style, that is all parameters are passed as read-only (direction = "in"), and the only modification or creation of objects is done by using the return type in an assignment operator. In describing interfaces, the adjective "this" indicates the entity whose object interfaces are being invoked. In OCL, this object is referred to as "self". If an object is passed as a parameter to the method of another object, it is referred to as a "passed" object.

### 5.1.4 Stereotypes

ED: all nonstandard stereotypes used in the UML model will be added to the list below.

Some entities in a UML model can be described by a "stereotype" which is included near the name of the object and enclosed in guillemets "«" and "»". The stereotype allows the model to extend UML to include descriptions of elements of the model. In this International Standard, in additions to the stereotypes define in UML 2, and in ISO 19103, the following stereotypes are also used:

- «union» – a type consisting of one and only one of several alternatives (listed as member attributes). This is similar to a discriminated union in many programming languages.
- «codelist» – similar to an enumeration, in that one of a number of values is possible, but differs in intent, in that a code list may be expanded over time.

Stereotypes are case insensitive, and different spellings are possible. «DataType», «Datatype» and «datatype» are all equivalent; the all lowercase «datatype» is preferred.

Most UML classifiers in this standard use the stereotype «interface». In the process of realization, the concrete implementation classes (or data structures) have the option to instantiate attributes either as data (attributes) or operations (derived attributes).

### 5.1.5 Data types and collection types

Several collection types are required to make the standard consistent, but these types do not have to be specific in terms of their interfaces. While these types are not included in UML, they are often implied by usage of the Object Constraint Language (OCL), see **ISO 19103**.

The most common of these interfaces is the finite set. If we have a type "T", we denote a new instantiated class type called "Set<T>" to consist of all finite, unordered sets of objects of type "T". Implementation environments often supply several common collection types such as arrays, and we do not wish to try to impose a universal interface on these types. **ISO 19103** includes an example interface definition for these types. This International Standard does not restrict the use of logically equivalent types native to particular implementation environments. Some basic class types and

© ISO 2013 – All rights reserved

parameterized types, such as these collections types that are used in this International Standard include the following:

- TransfiniteSet<T> – a possibly infinite set; restricted only to values. For example, the integers and the real numbers are transfinite sets. This is actually the usual definition of set in mathematics, but programming languages usually restrict the term set to mean finite set.
- Number, Float, Integer, Real – various simple value types usually instantiated as programming language primitives within the environment, see **ISO 19103**.
- Lengths, Area, Distance – various scalar values, associated to a particular unit of measure such as the meter or acre, see **ISO 19103**.

NOTE Most finite sets, lists, bags and references have been replaced by simple arrays in this standard. Since this standard uses «interface» classifiers in almost all cases, the distinction between these types is almost non-existent. Arrays come with a natural ordering, in cases where this ordering has a particular interpretation; it will be discussed in the associated text.

Representations of associations, in the "code listings", have used the convention most common in code generators for UML. Association roles have been used as member names of type T where T is the target «interface» class of the role name in the association. In cases where ambiguity could exist, the association name is used as the name space for the role <association_name>::<role_name>: <target_class>. Logically, this could also have been done by using the source class of the association, <source_class>::<role_name>: <target_class>. The semantics of a strong aggregation, one-way association is logically identical to a member attribute. One of the sources of alternative designs in UML is the use of associations versus the use of role-like attributes. Once a code generator has been used, the backward generation of UML association (lacking any other information) might round-trip engineer to pairs of role-like attributes.

Some data types are simply instances of the Record type defined in **ISO 19103** and in slightly different terms in ISO/IEC 11404. Since the latter International Standard has a specification that might be confused with parameter lists, this International Standard uses a slightly modified syntax ( " (.)" external parenthesis replaced by "<.>"):

```
Record Type:=="<" field-name ":" type[=default-value],… ">"
Record Instance:=="<" field-name ":" field-value,… ">"
```

Note That the syntax for a multiple return type is consistent with this syntax for Record, except that the braces are omitted. This International Standard uses the Record syntax above when it is stand-alone, but uses the standard UML multiple return type syntax when specifying operations that return record-like structures as anonymous types.

EMPTY (∅) refers to objects that can be interpreted as sets, and means that the set in question contains no elements. Unlike programming systems that have strongly typed aggregates, this International Standard uses the mathematical tautology that there is one and only one empty set and that any object representing that empty set is equivalent to any other set doing the same. The term NULL in English can mean empty, but in programming and database theory can mean "unknown" or "missing data." Care must be taken to distinguish NULL and EMPTY in geometry and topology. The empty set is not unknown, is it empty and more precisely known than most sets in which accuracy is concerned.

### 5.1.6 Strong substitutability

This International Standard assumes that implementation profiles and transfer schemas will be built using a strong version of substitutability. This means that at several places in designing an application schema, a profile builder may use a class in lieu of one defined in this schema as long as it supports the data, operation and associations required of the base class. The method of implementation of this substitutability is not normatively defined, and may be done in a variety of manners depending on the characteristics of the implementation environment. This is especially true of transfer standards, which by their nature depend on data types. Entities in transfer sets may only be tenuously related to the base class in this International Standard, in that they may be data-only


© ISO 2013 – All rights reserved

representational forms. Places where substitutability is most useful are examined in subclauses associated to the class most likely to take advantage of this technique.

This assumption requires a strict adherence to the semantics of subclassing as an "is type of" hierarchy. Each instance of a class must be a member of all of the sets defined by the semantics of the supertypes of that class. Thus, we can define a Circle to be a subtype of Ellipse, but not the other way around, even though this is counterintuitive to the notion that subtypes are more complex than their supertypes.

### 5.1.7 Naming conventions

Where possible, when an object class is to represent the common behavior of a set of defined things from Clause 4, the UML classifier will use the defined terms as its name. Since classifier names are capitalized and contain no space, and the defined term may contain several words, the classifier name will separate words using upper-camel-case concatenations (no spaces but each word beginning with a capital with all other letters in lowercase). Similarly the name may be some simplified key phrase. This "UpperCamelCase" rule is generally followed but may be violated if clarity or consistency with other standards is improved by minor violations of the rule. For example;

- Instances of direct positions will be represented by the datatype DirectPosition.
- Instances of geometry primitives will realize the interface GeometryPrimitive.
- Instances of points will realize the interface Point.
- Instances of curves will realize the interface Curve.
- Any class name from another standard will retain its originally format.

To keep track of the usage of other standards, a namespace-like prefix will often be used to indicate the source of an external classifier; for example "ISO19111::CRS" is a classifier defined in ***ISO 19111*** with a local name "CRS." Package names may be omitted if the target standard has only one classifier of the given name. Internal cross referenced classes might use the local requirements class name in a similar manner

The text may use either form of the name (defined or UML) and not have any change of meaning. Saying a classifier has a property simply means that each member of those realized classes have that property.

Property and operations names in the UML models may similarly use key phrases in lower-camel-case (same as upper-camel-case, but the first word begins in a lowercase letter). For example, the attributes/operations boundary, dimension, endPoint and startPoint are all used as names. Special operations («constructor») that create instance of the named classifier will use upper camel case to parallel the classifier name that it instantiates.

ED: Clarifications of which constructors may be used will be added, either here or in the particular interfaces affected. Separate but equivalent internal representations should support the same interfaces. Each valid internal representation could be used to support a constructor for classes supporting the interface.

## 5.2 Organization

The clauses in this document are organized in terms of UML packages. A package is a set of related types and interfaces that form a consistent component of a software system design. Packages do not usually form a complete system since they often invoke the services provided by other packages in the system. When one package, acting as a client, uses another package acting as a server, to supply

© ISO 2013 – All rights reserved

needed services, the client package is said to be dependent on the server package. This dependency occurs when an object class in the package accesses another object defined in the server package. Since it is rare in geographic information for geometry to be purely client or server, these stereotypes are not used in this International Standard. Since dependent classes are associated to their server via an association that can carry the request, most object class dependencies derive from object class associations. Each dependency between objects in different packages must be reflected by a package dependency. This package dependency is indicated in package diagrams using the graphic notation as in Figure 1.

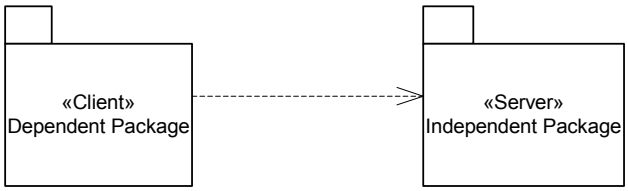



**Figure 1 — UML example package dependency**

Because of this client-server relation, inter-package dependencies define the criterion for viable application schemas. An application schema that contains an implementation of any package defined from this International Standard shall also contain implementations all of its dependencies.

NOTE Examples in the text are given where they are most appropriate to understanding the items in this International Standard, and, as such, often use implicit forward references to other items discussed later in the document. For example, the discussion of Envelope includes a forward reference to Line. In most cases, this is not confusing, since the item (type, operation, or attribute) of the forward reference is often semantically rich and corresponds closely to a commonly used term. If the reader finds this confusing, it is suggested that the entire document be read skipping the examples to establish an overview, and then reread carefully to include the examples.

## 5.3 Abbreviated terms (to be reviewed for additions and deletions before publish)

| | |
|---|---|
| 2D | 2-dimensional (meaning the geometric dimension of spatial portion of the CRS) |
| 3D | 3-dimensional (meaning the geometric dimension of spatial portion of the CRS) |
| AEC | Architecture, Engineering and Construction (graphics interface design systems) |
| API | Application program interface |
| ATS | Abstract test suite |
| C++ | Programming language based on C with object-oriented extensions |
| CAD | Computer Aided Design or Computer Aided Drafting (graphic interface design systems) |
| CRS | coordinate reference system |
| FoE | Figure of the Earth |
| LISP | Programming language based on LISt Processing. The standard is called Common LISP. |
| MBR | Minimum Bounding Region (sometimes rectangle) |
| OCL | Object Constraint Language |
| SQL 3 | Common name for SQL 99 during its development |
| SQL 99 | The SQL language specification adopted in 1999, which includes object-oriented data-type extension mechanisms |
| TIN | Triangulated Irregular Network |
| UML | Unified Modeling Language |
| URI | Uniform Resource Identifier (IETF RFC 3986) |

# 6 Coordinates and Core Geometry

## 6.1 Coordinate systems for Geometry — Semantics

### 6.1.1 Coordinates, points and locations

An n-dimensional coordinate space consists of all n-long arrays of numbers (or any type that can be interpreted as numbers); each array represents a point in the space. In particular situations, this may be restricted to a subset of such points, called the extent of validity, usually based on a set of constraints on values of the various offsets within the array. Each point is associated to a location,


© ISO 2013 – All rights reserved

but a single location may be the target of multiple coordinate space points. Locations given by such structures are called DirectPositions, see clause 6.2.6.

For example, a 1-dimensional coordinate space may be mapped to a circle by angular displacement in a chosen direction from a chosen origin on this circle. In this case, $(x)$ and $(x+2\pi)$ are distinct points in the coordinate space, but represent the same angular location on the circle. In this example,

$$location\big(DirectPosition(x)\big) = location\big(DirectPosition(x+2\pi)\big).$$

A coordinate space and a function that maps coordinate tuples to locations defines a coordinate system that is associated to an implementation of the interface DirectPosition that uses representations from this coordinate system.

All geometry objects in this standard are represented as constructions within a coordinate space that represent some set, usually infinite, of coordinate points, each point describable by a direct position. The simplest example would be a (coordinate) line defined by two coordinate points, $\overrightarrow{P_0}$ and $\overrightarrow{P_1}$ that represents all points in $\vec{X} = u\overrightarrow{P_1} + (1-u)\overrightarrow{P_0} \mid 0 \le u \le 1\}$.

Note This "line" in the coordinate system does not necessarily represent a line in real, physical world (i.e. a "line of sight" projected back down onto the surface of the earth, which is a segment of a geodesic curve (path of shortest length between any two of its points which are the "great circles" on a sphere).

Example A Mercator projection "line" is a line of constant bearing (rhumb line or loxodrome), and is either the arc of a circle (for east-west lines) or a segment of polar spiral. The only rhumb lines in a Mercator map that are geodesics are the equator or a line of constant longitude ("meridians") running north-south.

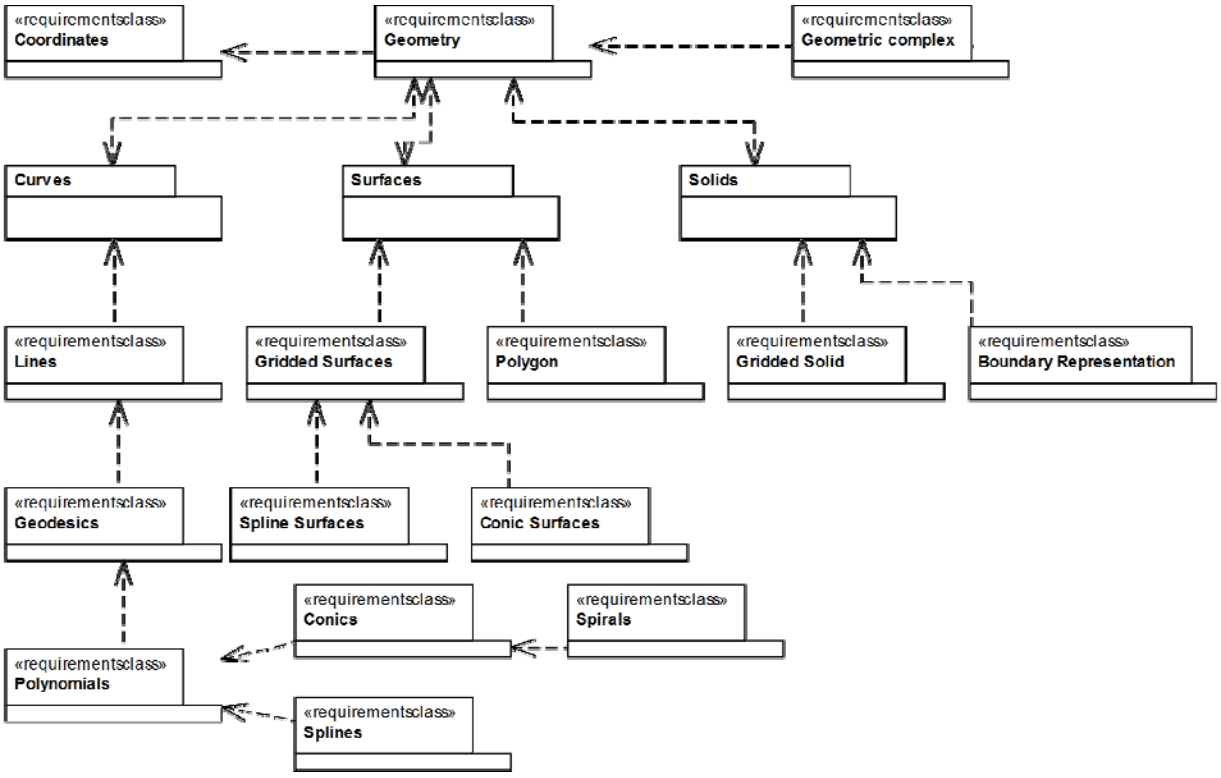



**Figure 2: — Geometry Packages and Requirement Classes**

© ISO 2013 – All rights reserved

### 6.1.2 Coordinates systems

Many of these spaces will be coordinate references systems as defined by ***ISO 19111***, but not all. None the less, all of these systems will satisfy some basic properties.

*In the system, each coordinate offset in the tuples shall contain instances of a class which can be converted into a single real number.*

This does not mean that the coordinate offset will be a real number, but that it can be converted to one. For example, "4° 30′" (four degrees, thirty minutes of arc) is equivalent to "4.5°" and could therefore be used as a coordinate offset. In the discussions in this standard, we will assume that all coordinates have been represented by real numbers, but this does not restrict representations such as DMS (degree, minutes, and seconds) as long as they can always be cast as a real number.

The coordinate offsets are not assumed to represent the same or even compatible units of measure. For example, in a 3D latitude-longitude system, the first two offsets are angles such as in degrees, and the third (usually elevation) will be a linear measurement such as meters. Therefore, the standard Euclidean distance formulae using the Pythagorean Theorem will not necessarily represent a proper distance in reality, but will define the same set of continuous functions or topology. In other words, the underlying space is a "manifold" and is locally a continuous image of the Cartesian coordinate space of the same dimension. The continuous function will not normally be an **isometry** (a preserver of distance, see the definition in clause 4).

Any function that depends on this sort of measure, such as distance, curve length or surface area, can have two interpretations: one using the local Euclidean metrics, which are essentially unit-less, and another which has been converted to an approximation of the real world measure. The handling of this is an application option, but should be done in a manner consistent with the conceptual model of the application.

*"Metric" operations in geometry-related interfaces shall use the best approximation of real-world values available.*

Almost all coordinate systems used in geography can be mapped back to an Earth-centered; Earth-fixed (ECEF) "geocentric" Cartesian coordinate system which is essentially a Euclidean space where the Pythagorean is valid. Mapping back to this datum coordinate system will always allow a final option for calculations using standard calculus techniques. Less extreme but equally valid is to map geometry to a local engineering system where the same "Euclidean-geometry" based calculations are feasible and have controllable error budgets.

*All locations in a list or array shall use the same coordinate system and shall reference reality in a manner representable by continuous functions from the coordinate tuples (DirectPositions) to reality in such a manner that "nearby" coordinates in the DirectPositions map to "nearby" positions in reality.*

This standard does not assume that these functions maintain topological dimension. For example, direct position may be given in "homogeneous coordinates" which equates, for every $w \neq 0$, the coordinate $(wx, wy, wz, w)$ in four-dimensional space with the point $(x, y, z)$ in three-dimensional space. This mapping is used heavily in projective geometry which aids implementers in creating

 © ISO 2013 – All rights reserved

realistic perspective views of higher dimensional geometry, and is used in this international standard in the definition of rational splines; by defining polynomial splines in homogeneous spaces.

Some offsets in the coordinate array of a direct position may have no spatial interpretation, such as a temporal, or one that corresponds to some other measurement associated to the position being described, such as a measurement in a linear reference system, or a physical property of space, such as temperature or pressure. Such offsets will not usually affect distance measures, unless those measures are interpreted by the application in some manner.

All direct position will have a projection into a pure spatial coordinate as would be associated to a coordinate reference systems (SC_CRS) from ISO 19111, or a location from ISO 6709. This projection is not assumed to be one to one, even if the coordinate system is purely spatial. It is assumed to be locally one to one, and bicontinuous. This means that the coordinate system will locally be reversible, but that is may wrap around singularities in the definition of the SC_CRS, such as occurs at 180°-longitude-line in a latitude-longitude system.

### 6.1.3 Metrics and distance measures

From the above description it can easily been seen that the "flat" concept of distance given by the Pythagorean Theorem is not directly applicable to DirectPositions except in limited circumstances.

The general solution is the use of Riemannian geometry, but this level of generality is not normally needed in geographic information. The key to the simplification is in **ISO 19111**, in that the datum surfaces that approximate the earth are usually embedded in a 3D geocentric Euclidean space. This means that the geometries on this datum surface are also in $\mathbb{E}^3$.

### 6.1.4 Underlying geometric space

Geometry systems inherently make assumptions about the "space" in which the geometric objects are defined. Classical Euclidean geometry makes the assumption that the objects line in an infinite flat plane represented, for example, by a Cartesian coordinate space with a standard Pythagorean metric. Geographic information if restricted to relatively small areas can be analyzed using classical Euclidean methods or algebraic calculations in a Cartesian coordinate space which often depend on the 'flat' Pythagorean metric. For larger areas, the effects of the earth's curvature become significant even for fairly primitive surveying systems.

Example: Much of the United States is surveyed using the **Land Ordinance of 1785**, which defines **Public Land Survey System (PLSS)** commonly called "the Jeffersonian Grid." In this system major boundaries are defined by "equally spaced" horizontal lines of constant latitude and vertical lines of constant longitude. Since these line are generally the boundary lines between owned parcels, roads often follow them. Since the lines are space by distances (6 or 24 miles), they misalign due to the curvature of the Earth, the north-south roads often "jog" (meaning: a brief abrupt change in direction) when they cross the major east-west lines also being used as rights-of-way for other roads.

To make such calculations more accurate, a more accurate model of the Earth's surface is created. The most common models involve 2D surfaces embedded in a 3D space. The 3D space is a Euclidean space, with a Pythagorean metric represented by Cartesian coordinates. The geometric/metric structure of these surfaces is inherited from the embedding 3D Cartesian space. On the surface, the distance between points is defined by curves of shortest distance lying on the surface that joins the points in question, similarly to the way lines in flat Euclidean spaces define distances in the Pythagorean metric. Thus, the distance between points on a sphere is defined by the length of the great circle on the sphere joining the two points. For ellipsoids, the geodesics are more complex, but they are defined as curves of shortest distances between points, and define the metric on the underlying surface.

© ISO 2013 – All rights reserved

They ones in most general use (in increasing order of accuracy) are:

- Map plane (a relatively local map projection onto a 2D plane)
- Sphere – a 2-sphere with a fixed radius, allowing the use of classical spherical geometry and trigonometry
- Ellipsoid – a bi-radial surface of revolution using an ellipse whose minor axis is aligned with the Earth's polar axis
- Geoid – an adjustment of an ellipsoid to an equipotential surface of gravity to adjust for the variations in Earth's local density and thus it gravity potential.

Issues of local usage can adjust CRS's to work particular well based ona local best fits of the FoE surface. For example, in the United States local mapping agencies often use State Plane Coordinate system specifically to preserve the accuracy of Euclidean geometry. To do this, there are 124 distinct state planes system use to cover the country. Such a system could use the "FoE — Plane" conformance class.

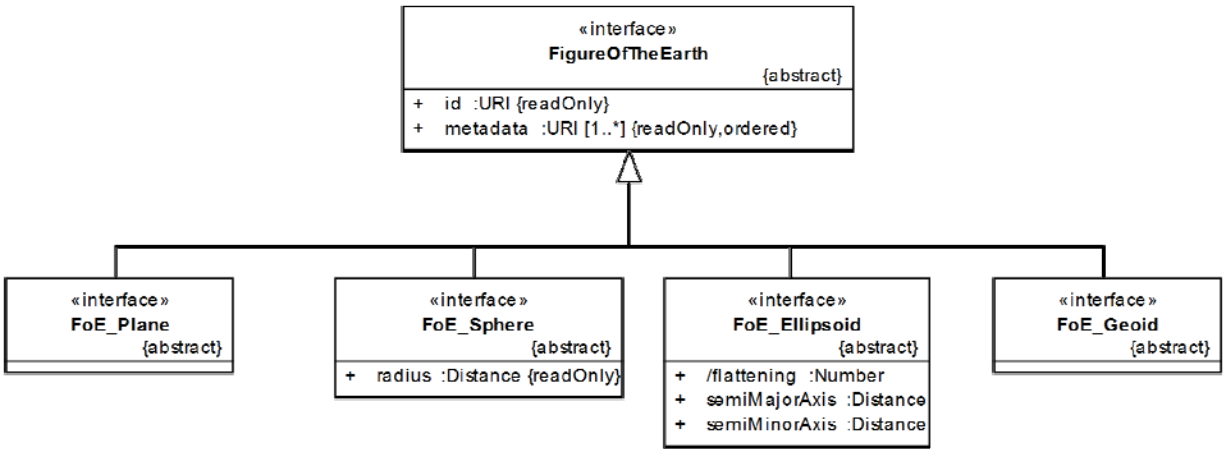



**Figure 3 — Figures of the Earth (FoE)**

Ed.: One of the major products of the consensus process is to determine requirements that become the definitions of the conformance classes. For conformance classes associated to liked-name UML «requirementsclass» packages, implementation of the contents of the package and of all of its dependencies will be a minimal set of requirements. Other requirements may be added to express aspects not readily obvious from the UML package contents.

## 6.2 Requirements class: Coordinate

### 6.2.1 Figure of the Earth (FoE)

#### 6.2.1.1 Interface: FigureOfTheEarth

A figure of the Earth is any surface used to describe the geometric space for determining various measures (principally length or area).

The interface FigureOfTheEarth has two attributes:

- id: URI — each figure of the earth has an unique identity. To support Web access, the identity is a URI.
- metadata: URI[1... *] — each figure should have a complete set of metadata


© ISO 2013 – All rights reserved

*An implementation of the interface Figure of the Earth (FoE) shall have properties specifying a unique identifier and the set of locations for associated metadata information that describe the representational surface of the earth; each represented by a URI (Universal Resource Identifier).*

*Any entity using coordinates to represent locations on the earth shall be associated to a coordinate reference systems (ISO 19111::CRS) and an earth model represented by a surface corresponding to a valid Figure of the Earth (FoE) .*

*There shall be a well-defined association between coordinates in that CRS and positions relative to that figure of the Earth.*

These three items, the CRS, the figure of the Earth and the association between coordinates in the CRS and position relative to that figure of the Earth are sufficient to define how geometric calculations such as length, distance and area may be done in data model using the geometry objects defined in this international standard.

*If the CRS is supplied but the Figure of the Earth is unspecified, the default surface used as the Figure of the Earth shall be the surface of the CRS object's related Datum as defined in ISO 19111.*

*When using a 2D CRS, all geometric measures, such as distance, bearing, length, area or volume, shall be consistent with the underlying Figure of the Earth or Datum.*

Note For more complex surfaces (ellipsoids and geoids) numeric approximations for distances and geodesic are often used. Implementations will be required to access these approximation to determine their error budget.

*For each 3D CRS, the horizontal 2D measures shall be consistent with the Figure of the Earth, and the vertical adjustment shall be consistent with an isometric embedding of the figure of the earth in 3D Euclidean space.*

*Elevations shall aways be determine by the directed distance from the Figure of the Earth to the position along the local upward normal of the FoE.*

*Each implementation system shall specify all CRS systems supported.*

In a 2D system, volume measures may be calculated from **footprints** (horizontal extent) and separate attribute information such as average height.

#### 6.2.1.2 Interface: FoE — Plane

Some geographic processing is still done using a "map metaphor" which essentially assumes that the flat image of an underlying graphic (paper or screen) differs from reality only by the scale of the map. This is especially useful in small areas where the curvature of the earth makes only small variations in the measures, in general areas of less than 10 kilometers (or 6 miles). In **ISO 19111**, these systems are called Engineering CRSs; and may be architectural or engineering drawings using large scales.

© ISO 2013 – All rights reserved

In general, the plane is mapped to the earth by assuming a single point (the "origin") of attachment (given in a geographic coordinate system) and all other points defined by measured offsets and directions (vectors) from this origin.

If the map is created by using local survey procedures, then one way to think of this may be to consider the map to be representation of the tangent plane at the origin, and the mapping to points on the map to be defined by the differential geometry "**exponential map**" from this tangent plane to any more accurate figure of the earth. The exponential map looks at the tangent plane in polar coordinates, being a bearing and a distance from the origin. This "polar" representation is then used to generate the geodesic on the earth with this initial bearing, and this length and using its end point as the position on the earth that is represented by the position on the map defined by this angle and this distance. Alternative manners for defining the position of the map positions' world points may use other map projections.

*The implementation shall support planar figures of the earth for it coordinate reference systems, and shall specify an error budget for metric calculations and an extent of validity for which that error budget is valid.*

If the plane is viewed as the x-y subspace of $\mathbb{E}^3$, then the normal to the plane is a vector parallel to the z-axis. The geodesics in the plane are the straight lines, but these can only be used with any accuracy in small areas since when they are brought back to the curved Earth, their geometry changes.

Note This essentially means that these entities use "map measures" and the CRS system in use is a projection.

#### 6.2.1.3 Interface: FoE — Sphere

The spherical approximation to the Earth's surface is one of the oldest in continuous use, and "central angle" latitude and longitude (valid for spheres but not ellipsoids) is the most common mental model of terrestrial locations. Further, the classical spherical trigonometry (2nd century AD) gives closed form solutions to most calculations needed for model-based measures.

Each sphere's shape is completely defined by its single attribute:

- radius: Distance — radius of the sphere (mean radius of Earth is about 6,371,009 m)

*The implementation shall support figures of the earth for it coordinate reference systems which are spheres in 3D Euclidean space of a fixed radius*

Assuming the sphere is centered in $\mathbb{E}^3$ at the origin $(x,y,z)=(0,0,0)$, then $x^2+y^2+z^2=r^2$ and the normals to the surface are the extensions of the radial lines from the center of the circle, i.e. the origin. The **geodesic** on the sphere are the great circles centered at the origin and creating circular curves centered there with their normals equal to the normals of the sphere.

Note This means in these additional CRS systems, 2D geodesics are great circles. The use of 3D adds offsets from the sphere and distances can be calculated by integrals in the embedding Euclidean space.

Math The planes that pass through the sphere's center cut the sphere in great circles. Since these circles are contained within these planes, their curvature vectors (2nd derivatives of the curves with respect to arc length) are vectors following the radial lines of these circles, and thus perpendicular to the sphere. Therefore, the great circles have 0 tangential curvature vectors and are thus geodesic**s** by the definition in clause 4.

 © ISO 2013 – All rights reserved

#### 6.2.1.4 Interface: FoE — Ellipsoid

Each ellipsoid's shape is completely defined by two attributes, it axes :

- semiMajorAxis: Distance — largest radius, anywhere along the equator ($a$)
- semiMinorAxis: Distance — smallest radius, at the two poles ($b$)
- inverseFlattening: Number — a single derived value equal to $\frac{a}{a-b}$, which is close to 300.0

*The implementation shall support figures of the earth for it coordinate reference systems which are an ellipsoid of rotation in 3D Euclidean space of a fixed equatorial radius and a potentially different polar axis radius.*

*Distance on ellipsoids shall be determined by geodesics on the ellipsoid which are curves with a zero tangential curvature vector.*

The normals of the ellipsoid are a little harder to visualize. Each point on the ellipsoid is on a unique ellipse perpendicular to the equatorial plane corresponding to a fixed longitude on either side of the polar axis. In the plane of this ellipse, there are two lines from this point, one to each of the foci of the ellipse (on the polar axis). The bisector of the angle between these two lines is normal to this ellipse, and the outward normal along this line is the normal to the elliptic curve (line of longitude). Since this ellipse is a "geodesic" on the ellipsoid, this same normal is the normal to the surface. A further description of the differential geometry is given in Clause 6.3.12.1 where the semantics of curves is discussed.

#### 6.2.1.5 Interface: FoE — Geoid

For the purposes of this standard, a geoid is any surface defined by offset mechanism from a standard ellipsoid, the most common example being an iso-surface of the gravity potential

*The implementation shall support figures of the earth for it coordinate reference systems which are geoids in 3D Euclidean space.*

### 6.2.2 Interface: ReferenceSystem

ReferenceSystem is an implementation the data type RS_ReferenceSystem from ***ISO 19111.*** The major difference is that the local versions uses a URI as an identity, since most useful geographic information has to be in a commonly understood coordinate system meaning it details needs to be easily available to all.

#### 6.2.2.1 Attribute: id: URI

The mechanism for this is a universally available identify for the system, which in current terms means some form of resource identity. Common practice for current implementation is a Uniform Resource Identifier that reference a common definition for the system in use.

*Each ReferenceSystem shall have a URI for identification.*

*URI used for this purpose should be HTTP-URIs.*

© ISO 2013 – All rights reserved

#### 6.2.2.2 Attribute: rsid: RSID [1...*]

In many applications, such as linear reference systems (ISO 19148) and moving features (ISO 19141), extra columns are added to the coordinates for "other" non-ISO 19111 purposes. The most common practice is to concatenate reference systems, which is easily done by listing an RSID for each component system.

*Each ReferenceSystem object shall have a sequence of RSID to identify projected reference system in it composition.*

It may be useful if commonly used compound reference systems have their own URI, but this an implementation detail and users are warned to take care in reading CRS and GeometricCoordinateSystem metadata

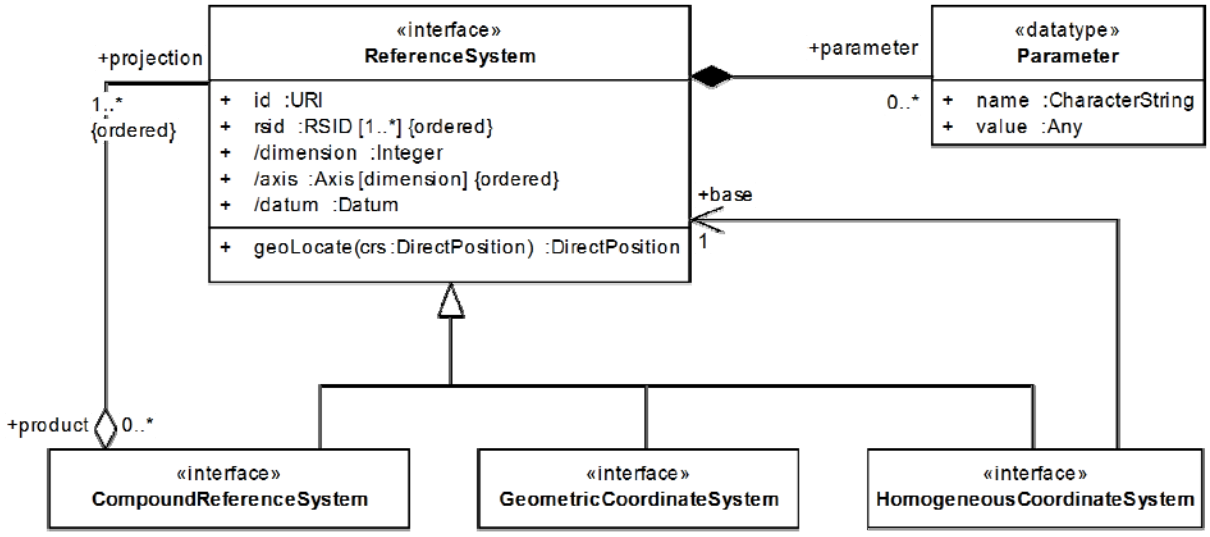



**Figure 4 — Reference systems**

#### 6.2.2.3 Derived attribute: dimension: Integer

The derived attribute dimension gives the number of offsets in the coordinate array which is the same as the number of axis in the system. The ordinates and the axes are in the same order, each ordinate offset corresponding to the same offset in the axis array.

#### 6.2.2.4 Derived attribute: axis: Axis [dimension]

The array axis lists the various axes from the identified reference systems (in the rsid array). Axis groups from each RSID are grouped in the order given in the corresponding reference system.

#### 6.2.2.5 Derived attribute: datum: Datum

The derived attribute datum is a reference to the geographic data used for the spatial components of the geometry in this system.


© ISO 2013 – All rights reserved

#### 6.2.2.6 Role: parameter: Parameter [0...*]

A ReferenceSystem can have direct connections to any number of parameters (name, value pairs) that may participate in the use of the system. Most parameters may be derived from the reference system ID's (RSID).

#### 6.2.2.7 Operation: geoLocate: DirectPosition

The operation geoLocate returns a pure spatial DirectPosition (in the ISO 19111 compliant SC_CRS, coordinate reference system).

### 6.2.3 Interface: CompoundReferenceSystem

#### 6.2.3.1 Semantics

A compound reference system allows the implementer of geometry entities to concatenate coordinate tuples into a single coordinate direct position values. Within a single set, all spatial coordinates should be from the same coordinate reference system, but since multiple Geometric coordinate systems can share the same spatial projection, the overall reference systems in a set would only have to share the spatial columns in the coordinate tuples.

*A Reference System with more than 1 projection shall support the interface for a CompoundReferenceSystem which has a derived attribute projection consisting of an array of RSID for each projection.*

If a reference system consists of a 2D CRS to establish horizontal positions, a 1D vertical reference system for elevation and a 1D temporal reference system for time, then the rsid will consist of an RSID identifier for the CRS, a second RSID identifier for the vertical system, and a third RSID for a temporal system.

A compound coordinate system is one which is composed of a sequence (represented by an ordered association role) of other coordinate systems. The dimension of a compound is the sum of the dimension of its projections.

#### 6.2.3.2 Role: projection: ReferenceSystem [1...*]

The ordered association role "projection" of CompoundCoordinateSystem lists the components of this instance of the interface, in the order that the ordinates are used in the compound coordinate system.

```
CompoundCoordianteSystem::projection:ReferenceSystem[1..*]
```

#### 6.2.3.3 Attribute: rsid: RSID [1...*]

The RSID identifiers of the ReferenceSystem instances in the projection role are repeated as an array of RSID system identifiers in the inherited attribute "rsid.".

Note The only reordering of axis or ordinates that is usable is in the essentially local-only GeometricCoordinateSystem which can rearrange the ordinates using its local attribute permutation.

© ISO 2013 – All rights reserved

### 6.2.4 Interface: HomogeneousCoordinateSystem

Homogeneous coordinates are useful in defining rational b-splines and are used in projective geometry.

*A HomogeneousCoordinateSystem shall be a GeometricCoordinateSystem which shall have an extra axis for a weight 'w.'*

See **Error! Reference source not found.** for a common representation of homogeneous coordinates. The usual mapping between the a 2D base coordinate system to the corresponding homogeneous system is $(x,y) \rightarrow (wx,wy,w); w \neq 0$, and so the usual projection back to the base system requires division by $w$ as in $(X,Y,W) \rightarrow (X/W, Y/\mathrm{W})$ and so $w$ must be non-zero. Alternately, some programs use $(x,y) \rightarrow (x,y,w); w \neq 0$, which makes projection back to the base system easier. Since all the "classes" in this international standard are "«interfaces»", the internal representation of the coordinates is an implementation decision.

*Homogeneous coordinates in encoding and transfer format should use the more projection-consistent encoding of* $(\mathbf{x},\mathbf{y}) \rightarrow (\mathbf{x},\mathbf{y},\mathbf{w}); \mathbf{w} \neq 0$ *to elimnate potential confusion.*

### 6.2.5 Interface: GeometricCoordinateSystem

#### 6.2.5.1 Semantics

The final step in creating a coordinate system for geometric calculation suitable for geographic measures is to combine the concepts in ISO 19111 in the CRS definitions, with the calculation mechanisms described above into a single interface.

*Every geometry used to represent a locus of positions in a geographic space shall be associated to a coordinate reference system (CRS) as defined in ISO 19111 or one its extensions, and to a Figure of the Earth which shall be used in all "metric" calculations implemented by its object representations.*

*If a geometry instance is associated to a CRS but not explicitly associated to a Figure of the Earth, then the implicit value of the figure of the earth for that geometry shall be the datum of the CRS.*

*If a geometry is associated to a figure of the earth different from the datum of the associated CRS, then the metadata associated to that geometry will make explicit reference to this inconsistency and will describe an error budget for errors introduce by this use of non-standard Figure of the Earth.*

Figure 5 describes the object model for these basic classes dealing with the relationship of geometry to its coordinate space. This should be considered as an information model, and any implementation of this model which contains the needed information, regardless of the form will be compliant.


© ISO 2013 – All rights reserved

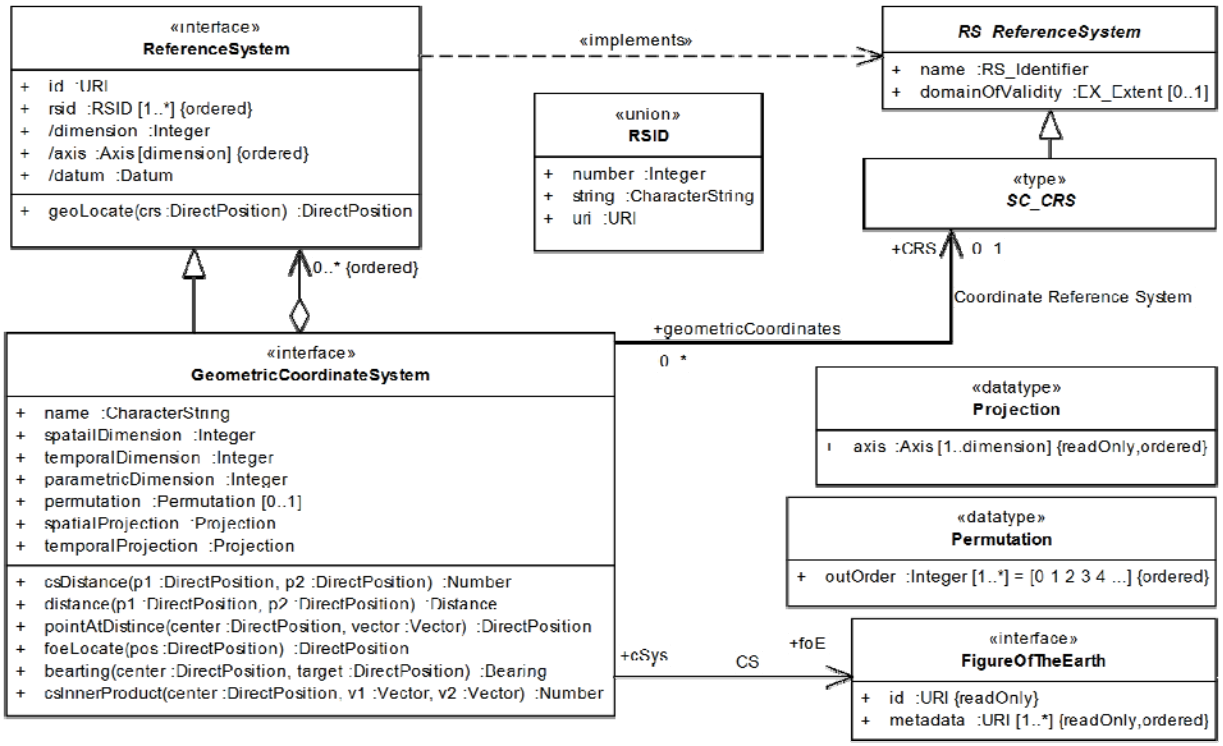



**Figure 5 — Reference System and CRS**

### 6.2.5.2 Attribute name: CharacterString

The attribute "name" is meant to be a human-readable name in the local language that is a mnemonic for the coordinate system.

```
GeometricCoordinateSystem::name:CharacterString
```

*Information communities should standardize the Geometric Coordinate System names for common usage and provide or specify metadata locations for all such systems.*

### 6.2.5.3 Attribute spatialDimension: Integer

The attribute "spatial dimension" indicates the dimension of the spatial coordinate systems in use. Usual values for the spatialDimension are 2 and 3. For spatial systems, the first two coordinates are usually "horizontal offsets." Examples include (latitude, longitude) in degrees and (easting, northing) in meters. The third dimension, at each point, is usually orthogonal, at least nominally, to the gravity iso-surfaces; i.e. the usual 3rd coordinate is some form of elevation, often but not always in meters.

```
GeometricCoordinateSystem::spatialDimension:Integer
```

*Only spatial dimensions shall be considered in measures of length, distance and area.*

### 6.2.5.4 Attribute temporalDimension: Integer

The attribute "temporal dimension" indicates the number of temporal axes in the coordinate system. In a standard spatio-temporal reference system, the temporal dimension is 1 and represents either an absolute position (implying both time and date) or the time passed since an event or time implied by

© ISO 2013 – All rights reserved

context. The temporal dimension may be larger, and the interpretation of each offset should be well-documented and defined in the metadata for the geometric coordinate system.

`GeometricCoordinateSystem::temporalDimension:Integer`

#### 6.2.5.5 Attribute parmetricDimension: Integer

The attribute "parametric dimension" is the count of the number of undifferentiated parameters stored in the geometric coordinate system. Other than being numeric in nature, there is no *a priori* restriction on parameters and their interpretations. Each offset should be well-documented/defined in the metadata for the geometric coordinate system.

`GeometricCoordinateSystem::parmetricDimension:Integer`

#### 6.2.5.6 Attribute permutation: Permutation [0.1]

The attribute "permutation" allows a local reordering of the coordinate offsets from the default one for the use of the local application. Permutations are represented by "0-up" ordinate offsets of the output. An empty permutation implies that the coordinates are not rearranged.

`GeometricCoordinateSystem::permutation:Permutation[0…1]`

Example A normally (latitude, longitude, elevation) system which is left-handed when subject to a [1, 0, 2] permutation becomes a (longitude, latitude, elevation) system, which is right-handed.

Note The most common purpose for use of an internal permutation is to maintain a "right-handed" coordinate system to allow standard mathematical calculation. For example, using Green's theorem to determine the orientation of a cycle $c(t)$ :

$$\left[\oint_c x dy - y dx > 0\right] \Rightarrow \left[c(t) \quad \text{is counter-clockwise}\right]$$

Is valid only if the (x, y) coordinate system is right-handed.

#### 6.2.5.7 Attribute spatialProjection: Projection

The attribute "spatialProjection" is a projection to a pure spatial coordinates system, equivalent to the class SC_CRS from ISO 19111. Projections are represented by ordinate offsets indexed by the non-negative integers {0, 1, 2 …} of the axes in the range of the projection function.

`GeometricCoordinateSystem::spatialProjection:Projection`

#### 6.2.5.8 Attribute temporalProjection: Projection

The attribute "temporalProjection" is a projection to a sequence of pure temporal coordinates system from ISO 19108. The semantics of multiple temporal offsets in a reference system should be covered in the geometric coordinate system metadata.

`GeometricCoordinateSystem::temporalProjection:Projection`


© ISO 2013 – All rights reserved

#### 6.2.5.9 Operation: csDistance (p1: DirectPosition, p1: DirectPosition): Number

The coordinate system distance is the normal distance function for the coordinates, using the appropriate Euclidean metric function (usually the Pythagorean formulae for a Cartesian system). This metric will normally produce the same topological structure on the Figure of the Earth as the next distance function. Because the units of the various axes may not be compatible, the result of this function does not normally correspond to a metric unit.

```
GeometricCoordinateSystem::
     csDistance(p1:DirectPosition, p1:DirectPosition):Number
```

#### 6.2.5.10 Operation: distance (p1: DirectPosition, p1: DirectPosition): Distance

The distance is the "real world" equivalent of the distance on the Figure of the Earth.

```
GeometricCoordinateSystem::
     distance(p1:DirectPosition, p1:DirectPosition):Distance
```

In a 2D system, the distance between two points is the length of the "shortest" geodesic on the Figure of the Earth joining them. Since most Figure of the Earth implementations use meters in their definitions, the most common units for the returned distance will be some multiple of meters, such as kilometers (based most likely on the accuracy of the systems and the resolution of the numbers in use).

#### 6.2.5.11 Operation: pointAtDistance (center: DirectPosition, vector: Vector): DirectPosition

The operation "point at a distance" solves the first geodetic problem which is given a starting point, an initial direction (bearing) and a distance; find a new point at that distance from the original point with that bearing as the initial tangent to the geodesic joining them. In the protocol for the operation, the parameter vector gives both the bearing (the direction of the vector) and the distance (the length of the vector).

Note Remember that the vector space at a point is a Euclidean space of the same dimension of the underlying manifold. In this case that is usually $\mathbb{E}^2$.

Semantically, the center point and the vector are an element of the tangent space on the Figure of the Earth at the center point. This operation is called the exponential map for the Figure of the Earth, mapping a Cartesian vector space (the tangent plane at the point) onto the surface. Locally, this maps a flat image of the locale of the center onto a geodesic surface (the Figure of the Earth).

Note The most common example of this usage are the Earth-fixed Engineering Systems (SC_EngineeringCRS) from *ISO 19111* most particularly those using polar coordinates, in which all locations are given coordinates based on physical measures (direction and distance) from a fixed point. Once the geographic location of the point is given, and the $\theta$ is aligned to a bearing, the exponential map becomes a CRS transform from the Engineering CRS to the coordinates systems using the Figure of the Earth as a datum.

This map projection from the local tangent space the Figure of the Earth is used in this international standard to create local equivalent instances of common Euclidean structures (such as circles) on the geographic Figure of the Earth space.

```
GeometricCoordinateSystem::
     pointAtDistance(center:DirectPosition,
     vector:Vector):DirectPosition
```

© ISO 2013 – All rights reserved

#### 6.2.5.12 Operation: geoLocate (pos: DirectPosition): DirectPosition

The geoLocate maps a direct position in the geometric coordinate space to the corresponding direction position in the geocentric coordinates of the embedding space for the Figure of the Earth. This function allows algorithms to map complex geographic geometries into classical 3D geometries in a flat 3D Euclidean space using a Cartesian coordinate system.

```
GeometricCoordinateSystem::
     geoLocate(pos:DirectPosition):DirectPosition
```

#### 6.2.5.13 Operation: bearing (center: DirectPosition, target: DirectPosition): Bearing

This operation "bearing" solves the second geodetic problem which is given two points; find the bearing of the geodesic that joins the first point to the second.

```
GeometricCoordinateSystem::
     bearing(center:DirectPosition,
     target:DirectPosition):Bearing
```

Note The solution to this problem is only guaranteed to be unique for a small distance. For example, if the two points are antipodal on a sphere, then the bearing between them in indeterminate and any bearing is valid. The only Figure of the Earth for which the solution is always unique is a plane; for example, a geometric coordinate system in a Mercator projection.

#### 6.2.5.14 Operation: csInnerProduct (center: DirectPosition, v1: Vector, v2: Vector): Number

The coordinate system inner product is a function that, for each tangent space defined by the center variable, defines a "dot product" for the local tangent space. The inner product is useful for doing vector measures such as length ($\langle \vec{x}, \vec{x} \rangle = \|\vec{x}\|^2$) and angle ($\langle \vec{x}, \vec{y} \rangle = \|\vec{x}\| \|\vec{y}\| \cos\theta$ where $\theta$ is the angle in the tangent space between $\vec{x}$ and $\vec{y}$).

```
GeometricCoordinateSystem::
     csInnerProduct(center:DirectPosition, v1:Vector,
     v2:Vector):Number
```

### 6.2.6 Datatype: DirectPosition

#### 6.2.6.1 Semantics

DirectPosition object data types (see Figure 6) hold the coordinates for a position within some coordinate system, which may be compound and represent an ordered sequence of other coordinate systems. Each coordinate system may be associated to at most one spatial coordinate reference system as described in ISO 19111.

NOTE Since DirectPosition is an interface, the actual implementation of the association to CoordinateSystem may be implemented in a variety of manners and may be derived in cases where the direct position is stored in a container having a similar relationship to a CoordinateSystem. If a DirectPosition is in one and only one container with such an association, it may derive its association to CoordinateSystem from this container.

 © ISO 2013 – All rights reserved

#### 6.2.6.2 Attribute: rsid: RSID [1...*] {ordered}

The attribute rsid is a sequence of reference system identifiers (RSID) of the various components of the coordinate system used for the direct position. Each RSID is a sequential subset of coordinate array for the axis in the corresponding system.

*The RSID should follow a general pattern of [spatial axes] ([temporal axes]) ([homogeneous Weight]) ([parameter axes])*

```
DirectPosition::rsid: RSID [1...*] {ordered}
```

#### 6.2.6.3 Attribute: /dimension: Integer

The derived attribute "dimension" is the dimension of the associated Reference System, and is thus the dimension of the position coordinate.

```
DirectPosition::dimension: Integer
```

#### 6.2.6.4 Attribute: coordinate: Number [*]

The attribute "coordinate" is a sequence of Numbers that hold the coordinate of this position in the specified reference systems.

```
DirectPosition::coordinate:Number[*]
```

#### 6.2.6.5 Coordinate System Role: referenceSystem: ReferenceSystem [RSID]

The derived association role "referenceSystem" is the sequence of coordinate systems, indexed by the RSID values in the rsid array from clause 6.2.6.2. Normally, the first in the array will be a realization of the type SC_CRS as described in *ISO 19111*.

```
DirectPosition::referenceSystem:RSID[1..*]
```

#### 6.2.6.6 Operation: spatialProjection (): DirectPosition

The operation "spatialProjection" shall project the DirectPosition to its pure spatial components. The resulting DirectPosition should be in an *ISO 19111* coordinate reference system.

```
DirectPosition::spatialProjection():DirectPosition
```

#### 6.2.6.7 Operation: pointProjection (): DirectPosition

The operation "pointProjection" shall project the DirectPosition to its pure spatial components, except that it shall still be in homogeneous form, it the original DirectPosition was. The resulting DirectPosition should be in an *ISO 19111* coordinate reference system.

```
DirectPosition::spatialProjection():DirectPosition
```

© ISO 2013 – All rights reserved

#### 6.2.6.8 Operations from Point

##### 6.2.6.8.1 Semantics

The following two operations are optional in that they are not needed for the principle purpose of DirectPosition, i.e. the indication of location. In a system that implements Point (see Clause 6.3.10), these functions could be replaced by casting the DirectPosition to Point and then executing them from the Point instance. Because these operations are valuable in navigating non-Euclidean Figures of the Earth (all but the planes), being able to bypass a Point instance constructor is worthwhile.

*Compliant implementations that support the interface Point, or do geometric analysis on and non-planar FoE, should support the operations "vectorToPoint" and "pointAtDistance" for DirectPositions.*

*Implementation standards dependent on this international standard may require these operations on direct positions.*

##### 6.2.6.8.2 Operation: vectorToPoint (toPoint: DirectPosition): Vector

The operation "vectorToPoint" will return a vector in the tangent space at the point whose direction determines a geodesic curve that intersects "toPoint" DirectPosition at a distance equal to the length of the vector. This operation solves the second geodetic problem.

```
DirectPosition::vectorToPoint (toPoint: DirectPosition): Vector
```

##### 6.2.6.8.3 Operation: pointAtDistance (bearing: Vector): DirectPosition

The operation "pointAtDistance" will return a DirectPosition given a vector in the tangent space at the point whose direction determines a geodesic curve that intersects that DirectPosition at a distance equal to the length of the vector. This operation solves the first geodetic problem.

```
DirectPosition:: pointAtDistance (bearing: Vector): DirectPosition
```


© ISO 2013 – All rights reserved

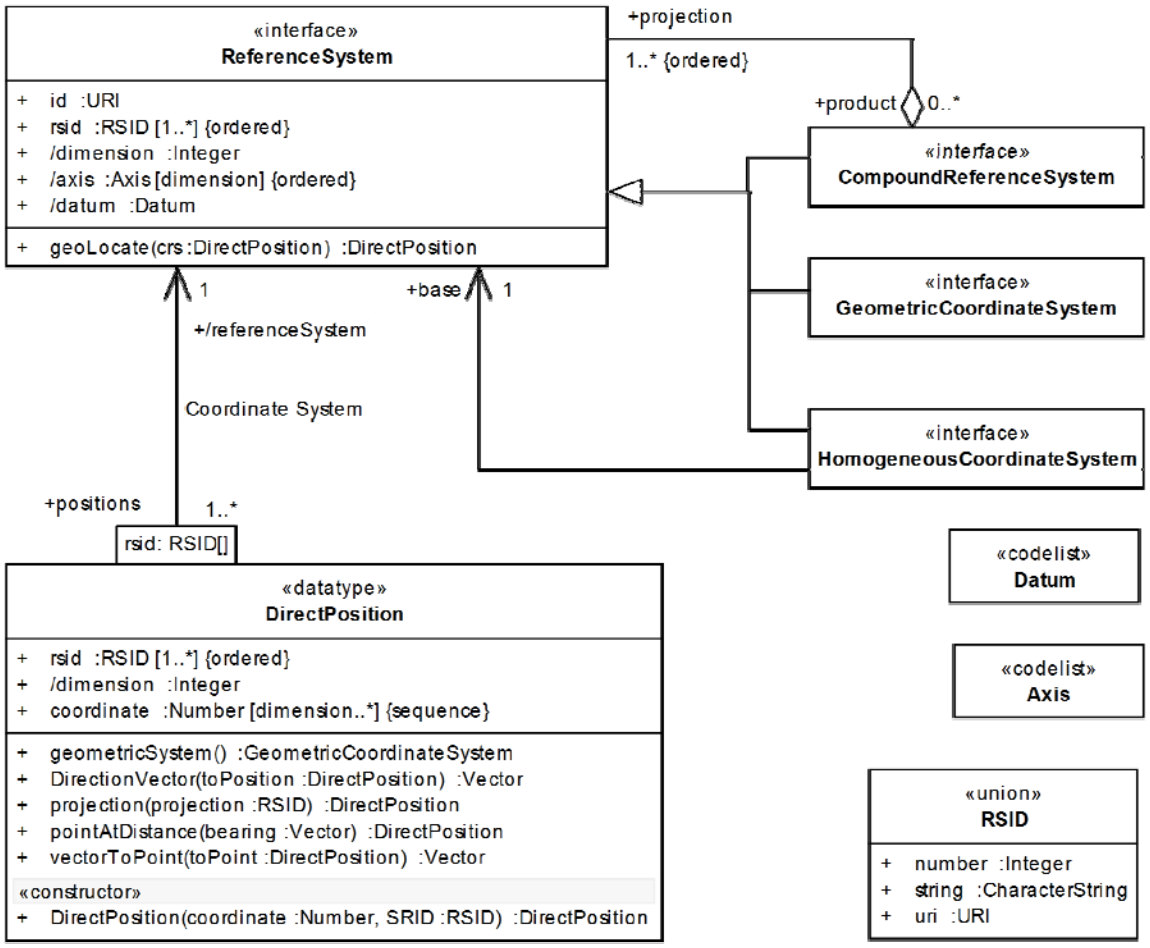



**Figure 6 — DirectPosition and ReferenceSystem**

### 6.2.7 Union, Datatype: RSID

The SRID, or spatial reference system identity, is an identifier for a reference system and a generic ID that can hold a variety of identity types for local use. The SRID (see Figure 6) content can be a number, a character string, or a URI. It is meant to be a local identity, but can be equal to global values from a recognized Authority.

*The use of an RSID equal to a global indentity linked to a recognized authority shall imply that the local object is consistent with the item defined by that recognized authrity.*

### 6.2.8 Codelist: Axis

#### 6.2.8.1 Semantics

The local instance of the Axis codelist (Figure 7) will contain the names of any axis that can be used in local instances of Reference System. Any name in this codelist will have a class of Axis Description that contains information on the axis' use.

*To keep axis types separate, the Axis code list may contain sub-list for each general type of axis, including the order: Spatial, Temporal and Parametric*

© ISO 2013 – All rights reserved

#### 6.2.8.2 Role: metadata: AxisDescription

Every axis is associated to a metadata description. The exact mechanism for this connection is an implementation option. The simplest mechanism is to use a standard name with a well-known definition (latitude, longitude) .

```
Derived attribute Axis::metadate: AxisDescription
```

### 6.2.9 Datatype: Axis Description

Each Axis type will have a metadata description (identified by the name of the Axis)'

*Each axis in use shall have an Axis Description associated to it by name. The Axis Decription instance shall have a URI pointer that identifies all information needed to use the axis in queation in a Geometric Coordinate System.*

### 6.2.10 Codelist: Spatial Axis

The SpatialAxis codelist will contain all axis names used to represent spatial extent where Euclidean geometry is useful, at least locally. Each name will be used in at least one of the SC_CRS instance in local use as defined in ISO 19111.

### 6.2.11 Codelist: Spherical Axis

The SphericalAxis codelist will contain all axis names used to represent spatial extent measured in angles, such as spherical or elliptical latitude or longitude, or a radial axis used in a polar coordinate system. Each name will be used in at least one of the SC_CRS instance in local use as defined in ISO 19111.

### 6.2.12 Codelist: Temporal Axis

The TemporalAxis codelist will contain all axis names used to represent temporal extent. Each name will be used in a temporal reference system in a TM_ReferenceSystem value ISO 19108 in local use .

### 6.2.13 Codelist: Parametric Axis

The ParametricAxis codelist will contain all axis names used to represent parameters. Each name will be used in one or more parametric reference system in local use as defined in **ISO 19111-2.**


© ISO 2013 – All rights reserved

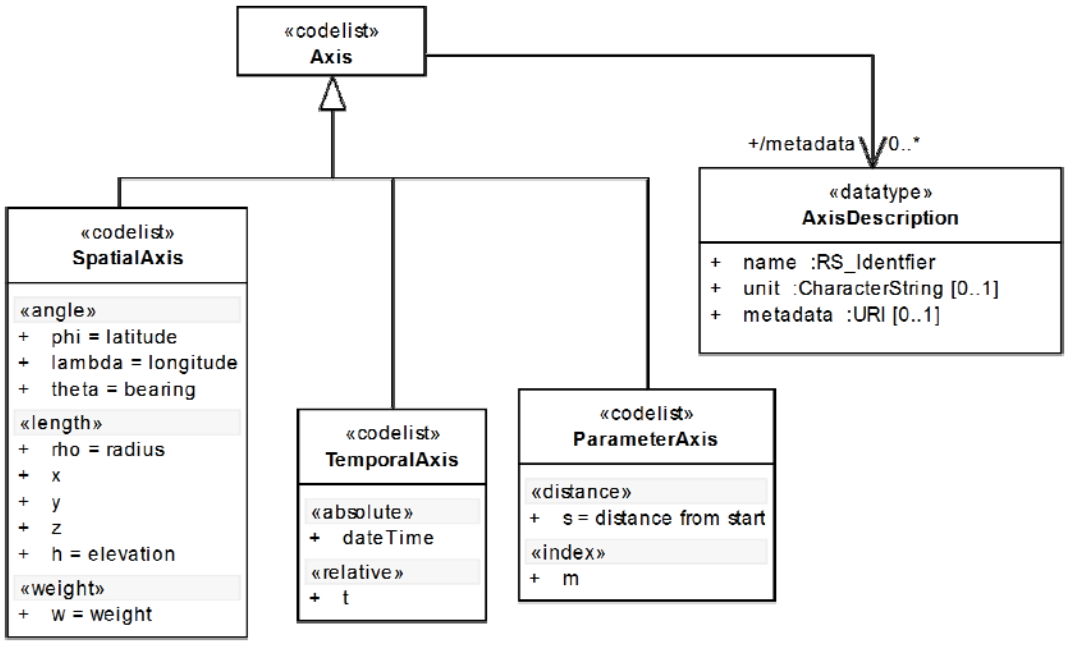



**Figure 7 — Axis**

### 6.2.14 Codelist: Datum

The local instances of the Datum codelist will contain the names or RS_Identifier values from Datums as defined in ISO 19111. Tracing the identifier name back to the data links the local datum to their definitions.

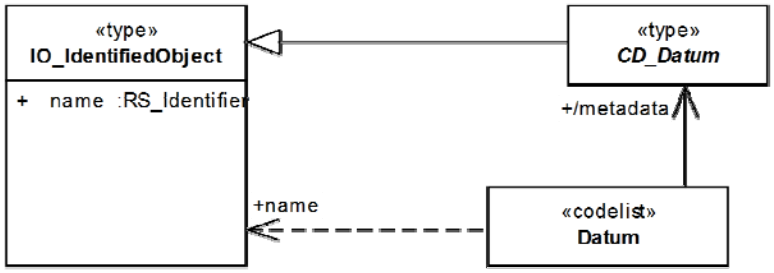



**Figure 8 — Datum**

### 6.2.15 Datatype: Parameter

#### 6.2.15.1 Semantics

The datatype Parameter can hold any parameter name and an assigned value for use in any Reference System.

#### 6.2.15.2 Attribute: name: CharacterString

The attribute “name” carries the name of the parameter.

```
Parameter::name: CharacterString
```

#### 6.2.15.3 Attribute: value: Any

The attribute “value” carries the value of the parameter.

© ISO 2013 – All rights reserved

```
Parameter:: value: Any
```

### 6.2.16 Datatype: Permutation

#### 6.2.16.1 Semantics

The datatype permutation holds the target positions of the input order. Positions are numbered zero-up; so an n-tuple in original order is (0, 1, 2…, n-1)

Example: If the columns of a 3-tuple are rearranged by 0→2, 1→0 and 2→1 (circular shift to the left); then the representation of the permutation is the sequence (1, 2, 0), i.e. (x, y, z)→ (y, z, x)

#### 6.2.16.2 Attribute: outOrder: Integer [1...*]

The outOrder attribute contains the target ordering by listing the output column by the offset for pre-image input column.

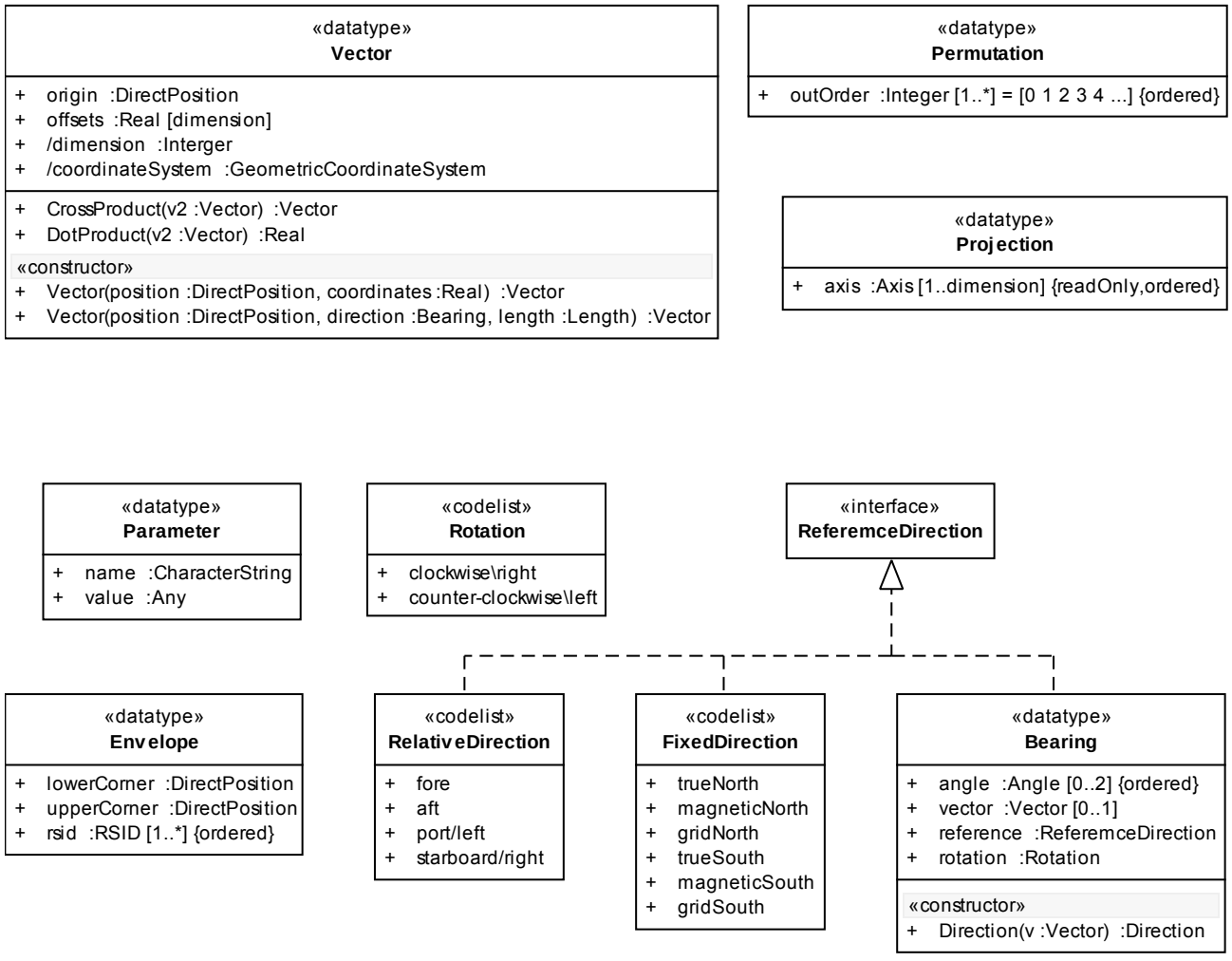



**Figure 9 — Vector, Bearing and Permutation**

### 6.2.17 Interface: ReferenceDirection

The reference direction interface is empty, but is to be "implemented" by any datatype that can represent a direction as a point. This lead to a circular, but valid, potentially recursive, definition for the datatype Bearing.


© ISO 2013 – All rights reserved

### 6.2.18 Datatype: Bearing

#### 6.2.18.1 Semantics

The discriminated union datatype Bearing denotes a direction. The bearing can take either one of two forms:

- A set of angles, one a planar bearing, and the second an measure of altitude (positive above the horizontal, negative below the horizon), essential creating a local spherical coordinate system.
- A directional vector from the SC-CRS at the initial point

Since the two types of measures are identical, the values of bearing are the same, but the interpretations will depend on the usage (usually attribute definitions).

The value of the datatype Bearing may be valid only at the point from which the measure is taken. The common "parallel" transformation of vectors from point to point is only valid if the Figure of the Earth in use is planar (i.e. Cartesian and hence Euclidean). The fundamental problem is that the sphere does not have a 2D universally valid global coordinate system for its tangent spaces, see **Error! Reference source not found.**. The use of a fixed direction reference such as "true north" does allow for some translation if the reference does exists and is unique at a position. For example, north is non-existent at the north pole, and not unique at the south pole.

*Bearings may be measured two ways, absolute (such as true north, see bearing) or relative (such as "fore," see relative bearing).*

*The reference direction for an instance of Bearing shall not transitively reference itself.*

Bearing is used to represent direction in the coordinate system. In a 2D coordinate reference system, this can be accomplished using a "angle measured from true north" or a 2D vector point in that direction. In a 3D coordinate reference system, two angles or any 3D vector is possible. If both a set of angles and a vector are given, then they shall be consistent with one another. In coordinate systems with higher dimensions, similar representations are possible.

#### 6.2.18.2 Attribute: angle: Angle [*]

In this variant of Bearing usually used for 2D coordinate systems, the first angle (azimuth) is measured from a coordinate axis (usually north) in a clockwise fashion parallel to the reference surface tangent plane. If two angles are given, the second angle (altitude) usually represents the angle above (for positive angles) or below (for negative angles) a local plane parallel to the tangent plane of the reference surface.

```
Bearing::angle:Angle[*]
```

#### 6.2.18.3 Attribute: direction: Vector [0, 1]

In this variant of Bearing usually used for 3D coordinate systems, the direction is express as an arbitrary vector, in the coordinate system.

```
Bearing::direction:Vector[0,1]
```

© ISO 2013 – All rights reserved

#### 6.2.18.4 Attribute: reference: ReferenceDirection

The attribute reference is the direction from which the bearing is measured, i.e. the direction of a positive measure. Most systems can use a negative number to express a measure opposite of the default rotation.

#### 6.2.18.5 Attribute: rotation: Rotation

The attribute rotation specifies the direction from which the bearing is measured.

### 6.2.19 Codelist Rotation

The codelist Rotation lists the two potential rotational direction for an angular measure

### 6.2.20 Codelist RelativeDirection

The codelist RelativeDirection lists the common potential relative reference directions for bearing, usually used in reference to a moving vehicle. The most common values include fore (forward), aft (backwards), port (90° left) or starboard (90° right).

### 6.2.21 Codelist FixedDirection

The codelist FixedDirection lists the common potential fixed reference directions for bearing, usually used in reference to the globe, a map, a coordinate system or a grid. The most common values include true north or south, magnetic north or south, grid north or south (reference to a map grid or projection).

### 6.2.22 Datatype Vector

#### 6.2.22.1 Semantics

The common "vector" in Euclidean spaces may be translated to any point in the space because the flat nature of the Euclidean space has a universal coordinate system that works at all "start" point of a vector.

The datatype vector in this standard must be associated to a point on the Figure of the Earth to be well defined. The directional parts of the vector must also specify the "start position" of the vector. Where the coordinate system if well-behaved, the spanning vectors of the tangent space are represented in the 3D geocentric space by the tangents to the coordinate functions. For example, in a latitude, longitude system, a local tangent space basis are the unit tangent vectors to the curves of constant longitude, and the curves of latitude, except for the poles where the longitude function has a zero tangent at the pole . If $(\text{lat}, \text{long}) = (\varphi, \lambda)$, then the $(d\varphi, d\lambda)$, the differentials of the two coordinate functions represented by unit tangents in the direction of increasing φ, and λ respectively, is a basis for the local tangent spaces.

#### 6.2.22.2 Attribute: origin: DirectPosition

The attribute origin is the location of the point on the Figure of the Earth for the vector is a tangent. The value is a direct position is associated to a coordinate system; this determines the system for the vector. The direct position spatial dimension determines the dimension of the vector.


© ISO 2013 – All rights reserved

#### 6.2.22.3 Attribute: offset: Real [dimension]

The attribute offset uses the coordinate system of the direct position and represents the local tangent vector in the differentials of the local coordinates.

Example If the Euclidean plane ($\mathbb{E}^2$) is in use, then the DirectPosition is a coordinate pair (x, y) and the vector is the differentials in both direction, hence offset is two long and has a coordinate base of (dx, dy).

#### 6.2.22.4 Derived Attribute: dimension: Integer

The derived attribute dimension is the dimension of the origin and therefore the dimension of the local tangent space of the vector.

#### 6.2.22.5 Derived Attribute: coordinateSystem: GeometricCoordinateSystem

The derived attribute coordinate system is the system of the origin and therefore determines the coordinates of the local tangent space in which the vector exists.

#### 6.2.22.6 Operations:
#### CrossProduct(v2:Vector):Vector
#### DotProduct(v2:Vector):Vector

*The two standard products of vector, the dot and cross produce shall be supported.*

The dot product returns a real value which is the sum of the products of the corresponding coefficients of the two vectors. The dot product is a 3rd vector perpendicular to the other two.

#### 6.2.22.7 Constructor:
#### Vector (position: DirectPosition, coordinates: Real[]): Vector
#### Vector (position: DirectPosition, direction: Bearing, length: Length): Vector.

The constructor vector creates a vector with the given offsets or the given direction and length, at the specified direct position, in the coordinate space of the direct position.

### 6.2.23 Interface: Envelope
#### 6.2.23.1 Semantics

Envelope is often referred to as a minimum bounding box or rectangle. Regardless of dimension, a Envelope can be represented without ambiguity as two DirectPositions (coordinate points). To encode a Envelope, it is sufficient to encode these two points. This is consistent with all of the coordinate systems in this standard. Recall, even if the CoordinateSystem is pure spatial but not globally one to one, the coordinate may not be valid in the associated SC_CRS.

#### 6.2.23.2 Attribute: Envelope::upperCorner

The “upperCorner” of an Envelope is a coordinate position consisting of all the maximal coordinates for each dimension for all points within the Envelope.

```
Envelope::upperCorner:DirectPosition
```

of that offset in all direct position in the original geometry.

© ISO 2013 – All rights reserved

#### 6.2.23.3 Attribute: Envelope::lowerCorner

The "lowerCorner" of an Envelope is a coordinate position consisting of all the minimal coordinates for each dimension for all points within the Envelope.

```
Envelope::lowerCorner:DirectPosition
```

Math: For each coordinate offset, the corresponding offset of the lower corner is the **Error! Reference source not found.** of that offset in all direct position in the original geometry and. the corresponding offset of the upper corner is the maximum of those same direct positions.

*The envelope of the empty geometry is by definition and shall be represented as the empty geometry.*

### 6.2.24 Engineering coordinate systems, Tangent spaces and local interpolations (informative)

In *ISO 19111*, the class CD_EngineeringDatum defines "the origins of an engineering coordinate reference system, and is used in a region around that origin." The class SC_EngineeringCRS is a "contextually local coordinate reference system associated with an engineering datum."

The tangent space at a point in a geometric coordinate system is the collection of all tangent vectors (of all lengths) at that point. The usual coordinate system used for vectors is a $\mathbb{E}^2$ or $\mathbb{E}^3$ (depending on the dimension of the underlying spatial CRS. The expression of bearings can be either polar $(r,\theta)$ or spherical $(r,\Omega,\theta)$ (see ISO 19111*)*. The operation pointAtDistance in the class GeometricCoordinateSystem, maps tangent vectors at a point to other points (given by a DirectPosition) using geodesic curves computation. This operation (formally called the exponential map in differential geometry) makes the tangent space (a copy of $\mathbb{E}^n$ a Cartesian coordinate space) equivalent to a SC_EngineeringCRS. Standard Euclidean constructions (lines, circles, ellipses, general conics and spirals) can work directly in this local engineering CRS and then projected onto the Figure of the Earth using the exponential map.

This international standard will recognize two general types of interpolations for geometry: algebraic and geodetic. Algebraic interpolations will treat the coordinates independent of geometric interpretation, strictly as variable in algebraic equations. Geodetic interpolations will use geometric considerations to interpolate direct position values, most commonly using the local tangent space at one of the control points to use Euclidean constructions and then using the exponential map to transfer these constructions to the FoE.

To avoid confusion, the names of algebraic interpolations will be distinct from those of geodetic interpolations. For example, 'linear' interpolation is an algebraic mechanism, linear interpolation in the tangent space generally produces geodesics originating from the control point and the mechanism will be called 'geodesic' interpolation.

## 6.3 Requirements class: Geometry

### 6.3.1 Semantics

The geometry packages contain the various classes for geometry. All of the geometry classes inherit, from the root class GeometryObject (Figure 10), an implicit link to a reference system (normally a geometric coordinate system) which defines the meaning of the coordinates in its direct positions.


© ISO 2013 – All rights reserved

Each primitive geometric object may contain other coordinate offsets, but will always have a constant coordinate system and FoE.

*All direct positions exposed through the geometry object interfaces shall be in the coordinate reference system of the geometric object accessed unless that operation protocol specifically specifies a different return reference system.*

*All elements of a geometric collection shall be available in the same coordinate reference system as specified by that collection.*

Figure 11 shows the basic behavioral classes defined in the geometry packages. Any object that inherits the semantics of the Geometry Object acts as a set of Direct Positions. Its behavior will be determined by which Direct Positions it contains.

Objects under geometry will be interpreted as metrically closed; any point whose distance to the geometry object is zero is on the object ("in" when the object is viewed as a set of direct positions). Curves will contain their end points; surfaces will contain their boundary curves, and solids will contain their bounding surfaces.

Note: The use on containment prepositions "in" or "on" for geometry is equivalent and may be used interchangeably. "On" is probably more appropriate when speaking of geometry as a location reference, but "in" sounds better when speaking of geometry objects as sets of positions.

When a primitive closes back onto itself, a curve starting and ending at the same point, or a surface which folds back onto itself and is thus on "both sides" of all of its boundary curves, it is called a cycle, and is normally the boundary of a geometry of one higher dimension.

*A geometric object shall be a combination of a coordinate geometry constructed from some number of DirectPositions and a coordinate system which defines the interpretation of the coordinate arrays in those DirectPositions.*

*In all of the operations, all geometric calculations shall be done in the coordinate system of the first geometric object accessed, which is normally the object whose operation is being invoked.*

*Returned objects shall be in the coordinate system in which the calculations are done unless explicitly stated otherwise by the semantics of the operation.*

Many of the protocols defined in this section are basically those of set theory. In general a geometric object can be viewed as a set (usually infinite) of geometric points, each representable by a DirectPosition (see 6.2.6).

*Object instantiations of geometric objects shall realize the GeometryObject interface. Object instantiations of geometric points, when used as values, shall realize DirectPosition.*

*GeometryObject and GeometryPrimitive are purely abstract in the sense that no object or data structure from an application schema can instantiate them directly.*

© ISO 2013 – All rights reserved

*Instances of GeometryObject and GeometryPrimitive shall also be instances of one of one of their dimension specific subtypes, such as Point, Curve, Surface or Solid.*

This is not the case for GeometryComplex, which can be directly realized by an application schema class. Although GeometryComplex is not explicitly implemented by this International Standard, it would be valid for an application schema to include a concrete class called "GeometryComplex" in a class library conformant to this International Standard. Recall that the name space of the application schema is different from that of this International Standard and such seemingly logical abuses of name are nonetheless valid. This is not the case for the purely abstract interfaces within this International Standard. These interfaces are logically incapable of supporting an implementation directly. Constructors on these interfaces result in instances of more concrete subtypes, not in direct logical instances of the root type.

### 6.3.2 Interface: GeometryObject

#### 6.3.2.1 Semantics

GeometryObject (Figure 10) is the root class of the geometric object taxonomy and supports interfaces common to all geographically referenced geometric objects. GeometryObject instances are sets of DirectPositions in a particular coordinate reference system. A GeometryObject can be regarded as a potentially infinite set of points that satisfies the set operation interfaces for a set of DirectPositions that is TransfiniteSet<DirectPosition>. Since an infinite collection class cannot be implemented directly, a Boolean test for inclusion shall be provided by the GeometryObject interface. This International Standard concentrates on vector geometry classes, but future work may use GeometryObject as a root class without modification.

Note: As a type, GeometryObject does not have a well-defined default state or value representation as a data type. Instantiated subclasses of GeometryObject will.

Attributes are values assigned to represent properties of an object. If the general content of the object is such that the value might be calculated from a reasonable representation of the object, this international standard has marked it as "derived." There is no functional difference between a derived attribute and an operation without parameters. In «interface» classifiers in UML, the mechanism of implementation is not restrictively defined.

*Implementations may support any attribute or operation from this standard in any manner, as derived or stored attributes or as functions with or without parameters.*

#### 6.3.2.2 Attribute: rsid: RSID [1...*]

```
GeometryObject::rsid:RSID[1…*]
```

The attribute "rsid" identifies the reference system used for all coordinates used within the geometry.

*All direct positions and geometry objects returned by accessing a geometric object will use the reference system identified by the "rsid" unless another RSID is implied or supplied by the interface.*

In some geometry types, additional information associated to direct positions may be required to calculate interpolations (e.g. homogeneous weights for rational splines). These additional values may be stored at the "end" of the direct position's coordinate array.


© ISO 2013 – All rights reserved

*If the geometry object inherits a reference system from a container, then the local reference system shall be consistent with the inherited system in that the rsid identifier of the container shall be the initial part of the rsid of the geometric object.*

Note This is often the logic for lists of parameters which can be extended in such a manner that new parameters are added on the "right" end of the parameter list.

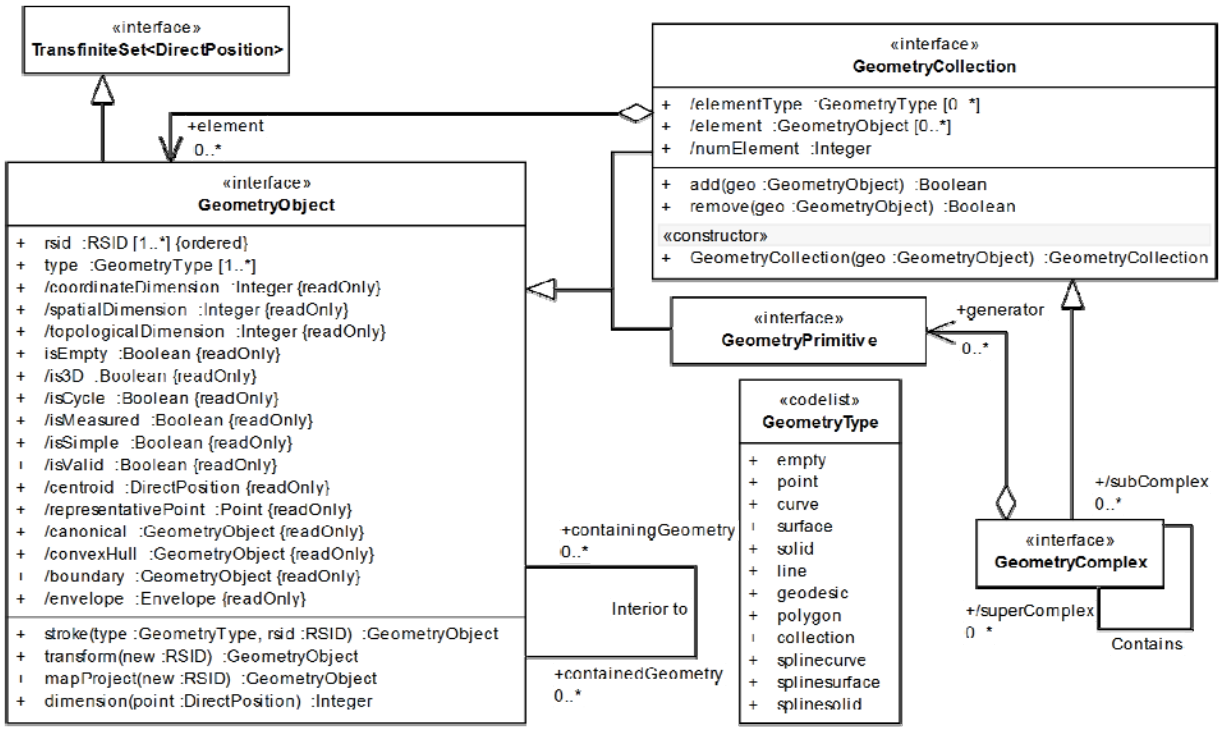



**Figure 10 — Geometry root object**

### 6.3.2.3 Attribute: type: GeometryType

The descriptions of geometric interpolations in the international standard are all defined in interfaces, not instantiable classes. The details of the instantiation of each type of interpolation may vary across implementation which is only obliged to follow the forms of the defined interfaces. The attribute type shall hold a reference back to the interface (or interfaces) from this international standard or its extensions that are supported by this particular instantiated object.

```
GeometryObject::type:GeometryType[1..*]
```

Example: A designed road bed may change interpolation mechanism, and may be presented as collections of reaches (road segments between intersections. Such a road may carry a type list include (curve, GeometryComplex)

*The type of the geometry object shall contain a value or values from the local GeometryType codelist which contains all instantiable types for geometry objects supported by the system.*

*The type value returned shall be the most specific types applicable to this particular instance of geometry object.*

© ISO 2013 – All rights reserved

#### 6.3.2.4 Derived attribute: coordinateDimension: Integer

The derived attribute coordinateDimension is the number of axis in the reference system.

#### 6.3.2.5 Derived attribute: spatialDimension: Integer

The derived attribute spatialDimension is the number of spatial axis in the reference system. The normal values are 2 or 3.

#### 6.3.2.6 Derived attribute: topologicalDimension: Integer

The derived attribute topologicalDimension is the largest topological dimension of the components of the object. For primitives, the topological dimensions are:

- Empty = -1-dimensional
- Point = 0-dimensional
- Curve = 1-dimensional
- Surface = 2-dimensional
- Solid = 3-dimensional

*The dimension of any collection shall be the largest dimension of a contained primitive.*

#### 6.3.2.7 Attribute: isEmpty: Boolean

The Boolean valued attribute isEmpty indicates that the instance of geometric object is the empty set. Since the empty set is unique, all empty objects are "spatially equal" to any other empty object. Once an instance is known to be empty (this.isEmpty = TRUE), the rest of the information in the object becomes redundant until the this.isEmpty Boolean is reset to FALSE.

*Implementation may use the isEmpty Boolean to set objects to NULL or Empty temporarily until a "transaction" or similar action is either completed or undone. Long term existence (persistence) of an "empty" geometry is not recommended but is not prohibited.*

Note: In this manner, "is.Empty" can be used much as "marked for deletion" is in some drafting systems, allowing a geometry to be kept for "undo" commands without its appearance in the display.

```
GeometryObject::isEmpty:Boolean
```

*Any geometric object for which isEmpty=TRUE, shall behave appropriately in its other operations and attributes; e.g., this.boundary, this.centroid and other derived GeometryObject are all empty, and any derived DirectPosition will return a NULL value or flag.*

*The topological dimension of an empty geometry shall be -1.*

There is only one empty geometry, for which topological dimension is undefined.


© ISO 2013 – All rights reserved

*All emty geometry objects shall be spatially equal.*

*If isEmpty=TRUE, the GeometryType="empty."*

#### 6.3.2.8 Derived attribute: is3D: Boolean

```
GeometryObject::is3D:Boolean
```

The Boolean attribute "is3D" indicates where the geometric object is using 3 spatial dimensions.

*If "is3D" is TRUE for an instance of Geometry Object, then the spatialDimension of that instance is 3.*

*The derived attribute is3D shall be true if and only if the coordinates' values in the geometry shall have a 3-dimensional spatial projection.*

#### 6.3.2.9 Derived attribute: isCycle: Boolean

```
GeometryObject::isCycle:Boolean
```

The Boolean attribute "isCycle" indicates whether the boundary is empty or not. The common term often used is "closed" but that has a specific topological meaning, and this standard uses "isCycle" to prevent confusion.

*The derive attribute "isCycle" shall be true if and only if the boundary of the geometric object shall have an empty boundary.*

The attribute "isCycle" will be TRUE if this GeometryObject has an empty boundary after topological simplification (removal of overlaps between components in non-structured aggregates, such as some subclasses of GeometryCollection). This condition is alternatively referred to as being "closed" as in a "closed curve." This creates some confusion since there are two distinct and incompatible definitions for the word "closed". The use of the word cycle is rarer (generally restricted to the field of algebraic topology), but leads to less confusion. Essentially, an object is a cycle if it is isomorphic to a geometric object that is the boundary of a region in some Euclidean space. Thus a point is a cycle as its boundary is empty. A curve is a cycle if it is isomorphic to a circle (has the same start and end point). A surface is a cycle if it is isomorphic to the surface of a sphere, or some torus (possibly with some number of holes – i.e. of a genus not 1). A solid with finite size in a coordinate space of dimension 3 is never a cycle. For a finite solid to be a cycle it has to exist in 4 dimensions.

EXAMPLE The following OCL uses the boundary to view a GeometryObject and then tests for an empty set using the isEmpty.

```
GeometryObject::isCycle()⇔ boundary.isEmpty
```

#### 6.3.2.10 Derived attribute: envelope: Envelope

The envelope of a geometric object is the smallest coordinate rectangle containing the geometric object.

© ISO 2013 – All rights reserved

*The derived attribute "envelope" shall return the smallest coordinate rectangle in the corresponding reference system that contains all the direct position in the geometry object.*

The "envelope" will be the minimum bounding box for this GeometryObject. This will be the coordinate region spanning the minimum and maximum value for each coordinate-offset taken on by DirectPositions in this GeometryObject. The simplest representation for an envelope consists of two DirectPositions, the first one containing all the minimums for each coordinate-offset, and second one containing all the maximums. However, there are cases for which these two positions would be outside the domain of validity of the object's coordinate system.

```
GeometryObject::envelope():Envelope
```

NOTE If a coordinate system "wraps" around singularities, there may be several alternate representations for the envelope. Applications that allow this will have to be careful in choosing one that makes sense in their operational concept. This international standard does not specify how this is done.

#### 6.3.2.11 Derived attribute: isMeasured: Boolean

The Boolean attribute "isMeasured" indicates where the geometric object is using a parameter axis for a measure dimensions (usually called "m").

*The derived attribute isMeasured shall be true if and only if the coordinates' values in the geometry shall have an axis for "m."*

#### 6.3.2.12 Derived attribute: isSimple: Boolean

```
GeometryObject::isSimple:Boolean
```

The Boolean attribute "isSimple" indicates whether the subject self-intersects (either by self-crossing or bine self-tangency).

*If a geometric object has a point of self-intersections or has a point of self-tangency then the derived attribute isSimple shall return FALSE. If the derived attribute isSimple returns FALSE, then geometric object has a point of self-intersection or has a point of self-tangency*

The attribute "isSimple" will be TRUE if this GeometryObject has no interior (non-boundary) points of self-intersection or self-tangency. In mathematical formalisms, this means that every point in the interior of the object must have a metric neighborhood whose intersection with the object is topologically isomorphic to an n-sphere, where n is the topological dimension of this GeometryObject.

Since most coordinate geometries are represented, either directly or indirectly by functions from regions in Euclidean space of their topological dimension, the easiest test for simplicity to require that a function exist that is 1-to-1 and bicontinuous (continuous in both directions). Such a function is a topological isomorphism. This test does not work for "closed" objects (that is, objects for which the isCycle operation returns TRUE) which will not be one-to-one on the boundaries of the parameterization domain.

 © ISO 2013 – All rights reserved

While Geometry Complexes shall contain only simple Geometry Objects, non-simple Geometry Objects are often used in "spaghetti" data sets.

NOTE "Spaghetti" is a pejorative (uncomplimentary) term, usually indicative of an unacceptable level of geometric anomalies and inconsistencies in the data that must be "cleaned" before use is made of it. Such inconsistencies can include (but are not limited to) any or all of the following anomaly types:

An undershot line is a line that should intersect another, but falls short leaving a small gap between it and the point of intersection. This is often hard to distinguish from real "near misses" between features (such as where a road is separated from another by a wall only one brick thick). This problem is especially difficult to handle when the undershoot fails to close a surface or polygon boundary. This is often indicative of the digitizer working at too small a scale and failing to "snap" to the end of lines.

An overshot line is a line that should intersect and terminate at another, but goes too far, leaving a small excess line on the far side of the point of intersection. This is often indicative of the digitizer working at too small a scale and trying to visually "snap" the end of lines.

End loop (a line that should intersect and terminate at another, but goes too far and then returns, leaving a small excess loop on the far side of the point of intersection. This is often indicative of the digitizer working at too small a scale and "snapping" a line after he has already overshot it.

Slivers and gaps are multiple lines that should represent the same geometry, but do not coincide, leaving areas of overlap between two surface boundaries (slivers), and gaps between them. This problem is particularly difficult to deal with in areas of braided streams, where the real geometry of the natural feature resembles the sliver and gaps of simple bad digitization practice. This is often indicative of multiple sources for the same data, which have been merged (but not properly conflated), into the same database.

he real problem with "spaghetti" comes in that the heuristics (either manual or automated) used to correct the problems often result in additional, but different factual errors. This can be a severe quality issue for geometry.

#### 6.3.2.13 Derived attribute: isValid: Boolean

The Boolean attribute "isValid" will initiate tests to assure that the local representation of geometry is valid.

*The derived attribute "isValid" shall be TRUE if and only if the structure of the geometric object is valid as geometry.*

#### 6.3.2.14 Derived attribute: centroid: DirectPosition

The attribute centroid: DirectPosition is a coordinate location that is the "center of gravity" of the geometry, see geometric centroid.

```
GeometryObject::centroid:DirectPosition
```

The centroid is a calculated value and may not be actually be interior to the geometric object.

*The derived attribute centroid shall return the position of the geometric centroid of the geometric object.*

For heterogeneous collections of primitives, the centroid only takes into account those of the largest dimension. For example, when calculating the centroid of surfaces, an average is taken weighted by area. Since curves have no area they do not contribute to this average.

```
GeometryObject::centroid():DirectPosition
```

© ISO 2013 – All rights reserved

Example: A "perfect" donut (torus) has its centroid in its "hole in the center".

NOTE There may be cases for which this position would be outside the domain of validity of the object's coordinate reference system, but this is unlikely, since the domain of validity of most coordinate reference systems is convex. If this unlikely case should arise the implementation shall decide on appropriate action.

Modifications of the centroid algorithm are possible to interpret curves as center-lines associated to some width, or points as centers with radius. In such cases, the dimensional restriction for mixed aggregates may not hold. The implementation specification shall decide on appropriate interpretations base on their requirements.

#### 6.3.2.15 Derived attribute: representativePoint: DirectPosition

`GeometryObject::representativePoint:DirectPosition`

The derive attribute representativePoint is a coordinate location that is somewhere on the geometric object.

*The derived attribute representativePoint shall return a position on the geometric object.*

The "representativePoint" may be implemented in different ways. It will be a point value (DirectPosition) that is guaranteed to be on this GeometryObject. The default logic may be to use the DirectPosition of the point returned by the operation "`GeometryObject::centroid`" if that point is contained in the object as a set of positions.

Another use of representativePoint may be for the placement of labels in systems based on graphic presentation. Definitions for symbology and type placement are outside the scope of this International Standard.

#### 6.3.2.16 Derived attribute: boundary: GeometryObject

The boundary attribute is the geometry representing the topological boundary of this GeometryObject. The boundary will always have a topological dimension one smaller than the current object, unless the object is empty or is a cycle ("`isCycle=TRUE`").

`GeometryObject::boundary:GeometryObject`

The finite set of Geometry Objects returned shall be in the same coordinate system as this GeometryObject. The boundary Geometry Objects returned may have been constructed in response to the operation.

*The return value of the boundary of a cycle ("isCycle=TRUE") shall always be an instance of the empty set as a geometry object.*

*The return value of the boundary of a point shall always be an instance of the empty set as a geometry object. For every instance of the point interface or of any representation of the empty geometry shall also have "isCycle=True."*

*The boundary of a curve if not empty shall be 2 points; the first (startPoint) and last (endPoint) point of the curve.*


© ISO 2013 – All rights reserved

*The boundary of a surface, if not empty, shall be a set of oriented curves, each of which is a cycle and is simple and each of which has the surface on its left, but not its right.*

*The boundary of a solid, if not empty, shall be a set of surfaces, each of which is a cycle.*

*In a 2D spatial system, the only surfaces which is a cycle (having a empty boundary) are the universal surface containing all possible positions or the empty set.*

*In a 3D spatial system, the only solids which is a cycle (having a empty boundary) are the universal solid covering all possible positions or the empty set.*

The organization of the set returned is dependent on the type of GeometryObject. Each instantiation of the subclasses of GeometryObject described below may specify the organization of its boundary set more completely dependent on the level to which the application has conformant implementations of GeometryComplex or TopologyObject.

The elements of a boundary shall be smaller in dimension than the original element. If the geometry is a non-empty, non-cycle, the dimension of the boundary is always one less than the original object.

*All objects in the boundary of a geometry object are of at least 1 dimension smaller than the dimension of the original object.*

*If a non-empty, non-cyclic geometry object is a primitive other than a point, then the boundary geometry object is 1 dimension smaller than the dimension of the object. The boundary of a finite set of points is empty.*

Note Some topology text use -1 as the dimension of the empty set to make points just another example of the above.

#### 6.3.2.17 Derived attribute: canonical: GeometryObject

```
GeometryObject::canonical:GeometryObject
```

The derived attribute canonical: GeometryObject is an alternate form of this object in canonical form which is equal to this object as a set of direct positions. A canonical form of a geometric object is a collection of simple primitives such that:

1. No primitive shares a point with the interior another primitive of any dimension and
2. There is no form satisfying this with fewer primitives (this implies that any two primitives of the same topological dimension sharing a portion of their common boundary have been merged into one primitive).

The first condition implies that the intersection of any two primitives is a subset of the intersection of their boundaries, regardless of dimension. In general, if two primitives of the same topological dimension share a portion of common boundary of 1 less dimension, then the two may be merged into one primitive, by adjusting orientation and simplifying the combined boundary. For example, if two curve meet at a point, in such a manner that the first curve ends at the point, and the second begins at the point, then the curves may be merged into a single curve that starts at the beginning of the first curve, go through the common boundary point, and then ends at the original second curve's end point. If two curves meet at a point, if needed one of them can be reversed to create the previous

© ISO 2013 – All rights reserved

case. If 3 curves meet at a point, then no merging is possible because of the 1st criteria above. Surfaces that meet along a common boundary curve can often be merged in a similar process, where the orientation of the curves is first matched, and then the common boundary is dissolved. A surface may in fact be merged with itself, as long as the resulting surface does not become a Möbius band (non-orientable surface).

The following are examples of canonical forms:

- A collection of distinct points
- A single, simple primitive.
- A collection of curves sharing only end points.
- A collection of surfaces sharing no common boundary curves (a common boundary curve would normally allow the surfaces on either side to be merged into a single surface)
- A collection of solids sharing no common boundary surfaces (same discussion as above)

It is not necessary to store geometry in canonical form, since its interaction with other geometries may make a more complex storage form more processing efficient, as is the case where an entire data set is topologically structured to optimize spatial query.

### 6.3.2.18 Derived attribute: convexHull: GeometryObject

The convex hull of a geometric object is the smallest convex set (represented as a geometry) containing the original object.

```
GeometryObject::convexHull:GeometryObject
```

*The derived attribute convexHull shall return the smallest convex set containing the geometric object.*

NOTE There may be cases for which this calculated convex GeometryObject would be partially outside the domain of validity of the object's coordinate reference system, but this is unlikely, since the domain of validity of most coordinate reference systems is convex. If this unlikely case should arise the implementation shall decide on appropriate action.

Convexity requires the use of "lines" or "curves of shortest length" and the use of different coordinate systems may result in different versions of the convex hull of an object. Each implementation shall decide on an appropriate solution to this ambiguity. For two reasonable coordinate systems, a convex hull of an object in one will be very closely approximated by the transformed image of the convex hull of the same object in the other.

### 6.3.2.19 Association Interior to: Role: containingGeometry: GeometryObject [0...*] Role: containedGeometry: GeometryObject [0...*]

In some cases, it may be difficult to determine if one geometry object is contained in another by a simple comparison of representation, usually when one of the geometry objects is a smaller dimension than that of the other. For example, a point on a surface may be subject to "round-off" errors which may make a coordinate calculation ambiguous. The purpose of the association "Interior to" is to make this relation explicit in those cases subject to ambiguity.

 © ISO 2013 – All rights reserved

#### 6.3.2.20 Operation stroke (type: GeometryType, rsid: RSID): GeometryObject

The operation stroke produces a "simpler" approximation of this geometric object. The parameter type is a set of acceptable primitives that may be used in the returned object. The optional parameter rsid may be used to specify the target reference system.

#### 6.3.2.21 Operation transform (new: RSID): GeometryObject

The operations transform returns a new geometric object, in the new reference system indicated by the passed RSID, equal to this geometric object within the accuracy of the transformation.

```
GeometryObject::transform(newCS:CoordinateSystem):GeometryObject
```

NOTE The behavior of a transform in non-spatial dimensions can be problematic. Normally, non-spatial coordinates will have their own internal logic for transformations to other compatible systems. If the GeometryObject is purely spatial, then this is transform is based on a change of coordinates from one SC_CRS to another.

#### 6.3.2.22 Operation mapProject (new: RSID): GeometryObject

The operations mapProject returns a new geometric object, in the new 2D spatial reference system indicated by the passed RSID, equal to this geometric object. The default value for the RSID will be the reference system using only the horizontal axes of the original reference system of this geometric object.

#### 6.3.2.23 Operation: dimension (point: DirectPosition = NULL): Integer

The operation "dimension" shall return the inherent topological dimension of this GeometryObject, which shall be less than or equal to the coordinate dimension. The dimension of a collection of geometric objects shall be the largest dimension of any of its pieces. Points are 0-dimensional, curves are 1-dimensional, surfaces are 2-dimensional, and solids are 3-dimensional. Locally, the dimension of a geometric object at a point is the dimension of a neighborhood of the point – that is the dimension of any coordinate neighborhood of the point. Dimension is unambiguously defined only for DirectPositions interior to this GeometryObject. If the passed DirectPosition is NULL, then the operation shall return the largest possible dimension for any DirectPosition in this GeometryObject's interior.

```
GeometryObject::dimension(point:DirectPosition=NULL):Integer
```

#### 6.3.2.24 Operations from TransfiniteSet realization

##### 6.3.2.24.1 Semantics

Each GeometryObject shall represent a potentially infinite set of points in its reference systems. This means that a GeometryObject acts as a generalized set in some calculations, and so is a subtype of the parameterized interface TransfiniteSet, where the type "T" is a realization of DirectPosition in the "coordinateSystem" of the GeometryObject. The GeometryObject realizes the following operations from the Interface TransfiniteSet<DirectPosition> (*ISO 19103*).

These operations differ a bit from "set" operations in that they are required to work with GeometryObjects which are always closed. Intersections or unions are not an issue, but differences tend to produce issues on boundaries of objects. All set-valued operations return closed versions of the sets as GeometryObjects.

© ISO 2013 – All rights reserved

#### 6.3.2.24.2 Operation: GeometryObject::contains(point: DirectPosition): Boolean
#### Operation: GeometryObject::contains(pointSet: GeometryObject): Boolean

The Boolean valued operation "contains" shall return TRUE if this GeometryObject contains another GeometryObject or a single point given by a coordinate (DirectPosition). The purpose of this operator is to instantiate TransfiniteSet<DirectPosition>::contains.

```
GeometryObject::contains(point:DirectPosition):Boolean // element
     of
```

The set theoretic definition for position contains (written $A \in B$) is fundamental to the definition of set.

```
GeometryObject::contains(pointSet:GeometryObject):Boolean //
     subset of
```

The set theoretic (point) contains is written $p \epsilon B$. The set theoretic definition for (set) contains (written $A \subseteq B$) is generalized up from the points of A, so:

$$\left[A \subseteq B\right] \Leftrightarrow \left[\left(a \in A\right) \Rightarrow \left(a \in B\right)\right]$$

If the given GeometryObject is a Point, then this operation is the equivalent of a set-element test for the DirectPosition of that point within this GeometryObject. Since point and other geometric objects share a common ancestor (GeometryObject), it is not normally necessary to differentiate between point containment and set containment for GeometryObject. The following OCL reiterates basic set theory axioms.

```
GeometryObject:
     contains(that:GeometryObject)and
     that.contains(other:GeometryObject)
     implies contains(other)
     contains(that:GeometryObject)and
     that.contains(p:DirectPosition)
     implies contains(p)
     contains(point:Point)implies contains(point.position)
```

NOTE "Contains" is strictly a set theoretic containment, and has no dimensionality constraint. In a GeometryComplex, no GeometryPrimitive will contain another unless a dimension is skipped.

#### 6.3.2.24.3 Operation: GeometryObject::intersects: Boolean

The Boolean valued operation "intersects" shall return TRUE if this GeometryObject intersects another GeometryObject. The set-theoretic definition of intersection (written $\mathrm{A} \cap \mathrm{B}$) is:

$$A, B \text{ are sets} \Rightarrow [\mathrm{A} \cap \mathrm{B} = \{\mathrm{x} \mid \mathrm{x} \in \mathrm{A} \wedge \mathrm{x} \in \mathrm{B}\}]$$

Two geometry objects intersect if their intersection is not empty.

$$A \text{ intersects } B \Leftrightarrow \mathrm{A} \cap \mathrm{B} \neq \varnothing$$


© ISO 2013 – All rights reserved

The purpose of this operator is to instantiate TransfiniteSet<DirectPosition>:: intersects.

```
GeometryObject::intersects(pointSet:GeometryObject):Boolean
```

#### 6.3.2.24.4 Operation: GeometryObject::equals: Boolean

The Boolean valued operation "equals" shall return TRUE if this GeometryObject is equal to another GeometryObject. The set-theoretic definition of equal (written $A = B$) is:

$$A,\ B\ are\ sets \Rightarrow \left[\left[A = B\right] \Leftrightarrow \left[x \in A \Leftrightarrow x \in B\right]\right]$$

Or

$$A,\ B\ are\ sets \Rightarrow \left[\left[A = B\right] \Leftrightarrow \left[B \subseteq A\right] \wedge \left[A \subseteq B\right]\right]$$

The purpose of this operator is to instantiate TransfiniteSet<DirectPosition>::equals.

```
GeometryObject::equals(pointSet:GeometryObject):Boolean
```

Two different GeometryObjects are equal if they return the same Boolean value for the operation GeometryObject::contains for every DirectPosition within the valid range of the coordinate reference system associated to the object.

NOTE Since an infinite set of DirectPositions cannot be tested, the internal implementation of equal must test for equivalence between two, possibly quite different, representations. This test may be limited to the resolution of the coordinate system or the accuracy of the data. Application schemas may define a tolerance that returns true if the two GeometryObjects have the same dimension and each direct position in this GeometryObject is within a tolerance distance of a direct position in the given GeometryObject and vice versa.

#### 6.3.2.24.5 Operation: GeometryObject::union: GeometryObject

*The "union" operation shall return the set theoretic union of this GeometryObject and the passed GeometryObject. The set-theoretic definition of difference (written $A \cup B$ ) is:*

$$A, B\ are\ sets \Rightarrow \left[A \cup B = \left\{a \mid a \in A \vee a \in B\right\}\right]$$

The purpose of union is to instantiate TransfiniteSet<DirectPosition>::union.

```
GeometryObject::union(pointSet:GeometryObject):GeometryObject
```

#### 6.3.2.24.6 Operation: GeometryObject::intersection: GeometryObject

The "intersection" operation shall return the set theoretic intersection of this GeometryObject and the passed GeometryObject. The set-theoretic definition of intersection (written $A \cap B$) is:

$$A,\ B\ are\ sets \Rightarrow [A \cap B = \{a \mid a \in A \wedge\ a \in B\}]$$

The purpose of intersection is to instantiate TransfiniteSet<DirectPosition>:: intersection.

© ISO 2013 – All rights reserved

```
GeometryObject::intersection(pointSet:GeometryObject):GeometryObje
     ct[*]
```

#### 6.3.2.24.7 Operation: GeometryObject::difference: GeometryObject

The "difference" operation shall return the set theoretic difference of this GeometryObject and the passed GeometryObject. The set-theoretic definition of difference (written $A - B$ or $A \setminus B$) is:

$$A,\ B \text{ are sets} \Rightarrow [\mathrm{A} - \mathrm{B} = \{a \mid a \in A \wedge\ a \notin B\}]$$

The purpose of difference is to instantiate TransfiniteSet<DirectPosition>::difference.

```
GeometryObject::difference(pointSet:GeometryObject):GeometryObject
```

NOTE The difference operation is not symmetric and A.difference (B) is usually not the same as B.difference (A).

#### 6.3.2.24.8 Operation: GeometryObject::symmetricDifference: GeometryObject

The "symmetricDifference" operation shall return the set theoretic symmetricDifference of this GeometryObject and the passed GeometryObject. The set-theoretic definition of symmetric difference or "exclusive or" difference ( $\mathrm{A} \oplus \mathrm{B}$, $\mathrm{A} + \mathrm{B}$, $\mathrm{A} \Delta \mathrm{B}$ or $\mathrm{A} \nabla \mathrm{B}$; $\mathrm{A} \oplus \mathrm{B}$ preferred) is:

$$A,\ B \text{ are sets} \Rightarrow \left[ A \oplus B = \left(A - B\right) \cup \left(B - A\right) = \left(\mathrm{A} \cup \mathrm{B}\right) - \left(\mathrm{A} \cap \mathrm{B}\right) \right]$$

The purpose of symmetricDifference is to instantiate TransfiniteSet<DirectPosition>:: symmetricDifference.

```
GeometryObject::symmetricDifference(set:GeometryObject):GeometryOb
     ject
```

### 6.3.3 CodeList: GeometryType

The code list GeometryType is a local list of types inheriting from GeometryObject. It essentially represents the software contract between the implementation and its users for the support of types of geometry, based on dimension and mechanism of interpolation.

The only two types that are always required to be in the list are "empty" and "point" as no logically consistent geometry system can operate without them.

### 6.3.4 Interface: MeasuredGeometry

The MeasuredGeometry (see Figure 11) extension of the GeometryObject base class is used for any reference system that supports a measure axis. The most common usage of a measure is for curves, and the geometry return type will usually be either be a Point, a Curve or Null.

#### 6.3.4.1 Operation: MeasuredGeometry:: locateAlong (mValue: Number): GeometryObject

The operation locateAlong creates a new geometry where the m-value is the passed number.


© ISO 2013 – All rights reserved

#### 6.3.4.2 Operation: MeasuredGeometry:: locateBetween (mStart: Number, mEnd: Number): GeometryObject

The operation locateBetween creates a new geometry where the m-value is in the closed interval given by the passed numbers, mStart and mEnd.

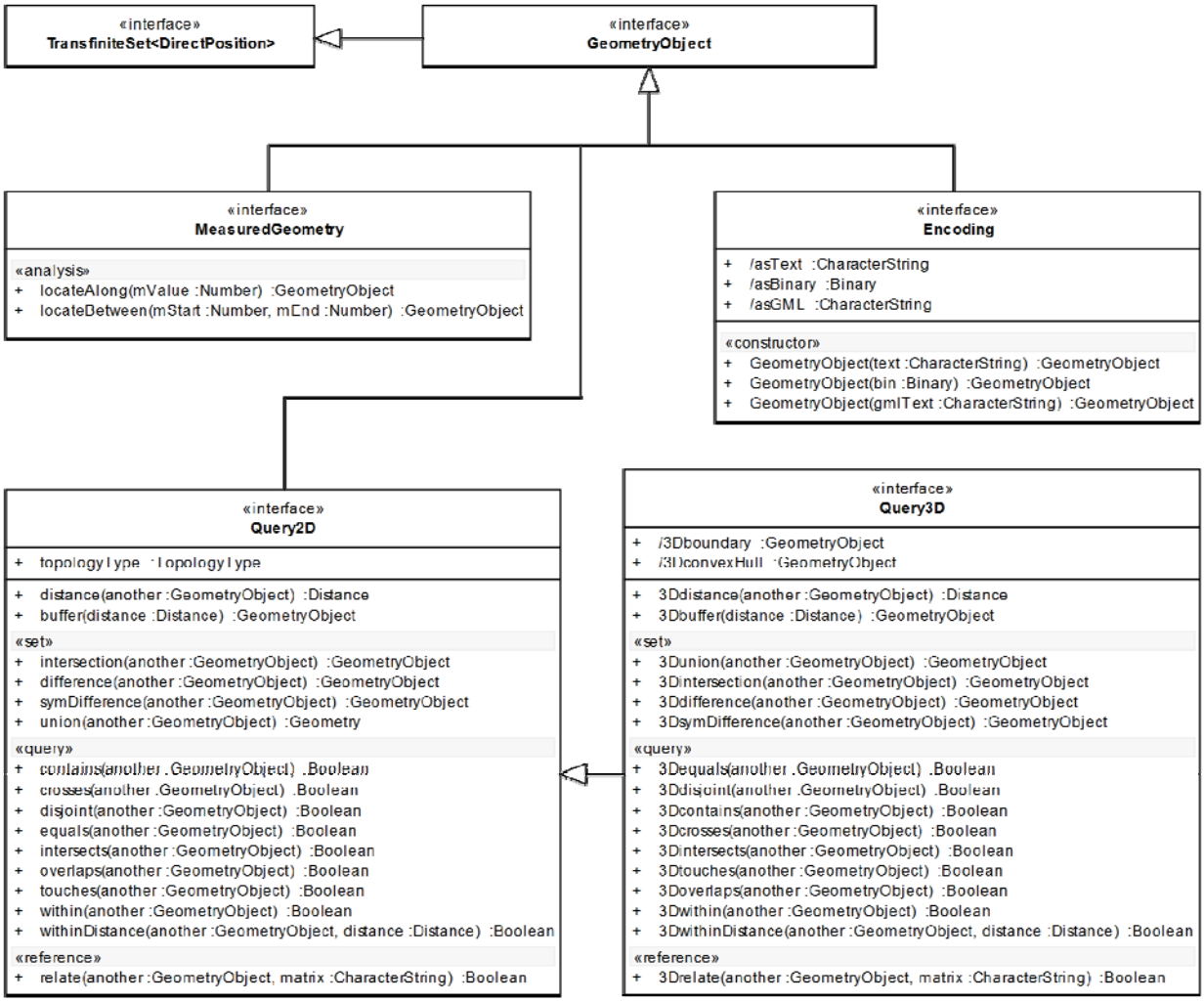



**Figure 11 — Geometry Root Extensions**

### 6.3.5 Interface: Encoding

#### 6.3.5.1 Semantics

The Encoding sub-interface for GeometryObject supplies a common set of operations for moving between text or binary representations of Geometry and object instantiations. The three common alternative representations are:

1. GML (any version may be supported)
2. WKT (well-known text)
3. WKB (well-known binary)

#### 6.3.5.2 Derived Attribute: asText: CharacterString

The attribute asText returns a WKT character string that is a representation of this geometry object.

© ISO 2013 – All rights reserved

#### 6.3.5.3 Derived Attribute: asBinary: Binary

The attribute asBinary returns a Bit string that is a representation of this geometry object.

#### 6.3.5.4 Derived Attribute: asGML: CharacterString

The attribute asText returns a GML character string (an XML fragment) that is a representation of this geometry object.

#### 6.3.5.5 Constructor: GeometryObject (text: CharacterString): GeometryObject

The constructors GeometryObject takes a WKT character string and instantiates a logically equivalent geometry object.

#### 6.3.5.6 Constructor: GeometryObject (Binary: Binary): GeometryObject

The constructors GeometryObject takes a Bit string in WKB and instantiates a logically equivalent geometry object.

#### 6.3.5.7 Constructor: GeometryObject (gmlText: CharacterString): GeometryObject

The constructors GeometryObject takes a GML (XML) element and instantiates a logically equivalent geometry object.

### 6.3.6 Interface: Query2D

#### 6.3.6.1 Semantics

The Query2D sub-interface of Geometry Object supplies interfaces for query in 2D coordinate spaces or in 3D coordinate spaces where the geometry objects are projected on the horizontal surface (the figure of the Earth in use). For each of the operations in this interface, each of the the geometric objects (if not already 2D) is projected onto the FoE surface being used in the geometric coordinate system.

#### 6.3.6.2 Attribute: topologyType: TopologyType

The attribute topologyType names the type of topology assumed for the query operators. This name will indicate how for each geometry object the spatial extent is divided into “interior, boundary and exterior.” These sets define the mechanism for the topological operators (non-metric) with this object.

#### 6.3.6.3 Operation: distance (another: GeometryObject): Distance

The operations distance returns the distance between the input geometric object and this geometric object after they have been projected onto the Figure of the Earth being used.

This distance is the minimal distance between two points in the two projected objects. If the objects overlap, then the distance is zero.

This distance is defined to be the greatest lower bound of the set of distances between all pairs of points that include one each from each of the two geometry objects. If necessary, the second geometric object shall be transformed into the same coordinate system as the first before the distance is calculated.


© ISO 2013 – All rights reserved

```
distance(geometry:GeometryObject):Distance
```

The return value may be in one of the units of measure data types defined in ISO TS 19103.

#### 6.3.6.4 Operation: buffer (distance: Distance): GeometryObject

The buffer operation creates a new geometry object that contains all direct positions that are within a distance (less than or equal to the distance) of this geometry object.

Note It is necessary to assure that geometry objects are topologically closed (by definition). Since the distinction between less than a distance or equal to a distance is beyond the capability of any digital computation, this should not matter since the inherent limitation of digital computations makes distinguishing between less than or equal to a distance.

The distance operation returns the distance between the parameter geometry object and this geometry object. The distance will be consistent with the figure of the earth in use, and the distance on the FoE as is consistent with the 2D nature of this interface (Query2D). Thus, the operations buffer creates a new geometric object containing all the direct positions whose distance from this geometric object is less than the radius passed. If the radius is "0", the resulting object is equal to this geometric object. If the radius is negative, the returned geometric object contains all directions positions within this object whose distance from the boundary is greater than the absolute value of the passed radius. The GeometryObject returned is in the same reference system as this original GeometryObject. The topological dimension of the returned GeometryObject is normally the same as the spatial dimension of the reference system - a collection of Surfaces in 2D space and a collection of Solids in 3D space, but this may be application defined.

```
buffer(radius:Real):GeometryObject
```

NOTE There are cases for which this GeometryObject would be partially outside the domain of validity of the object's reference system. If this case should arise the implementation shall decide on appropriate action.

#### 6.3.6.5 Set operation: intersection (another: GeometryObject): GeometryObject

The operation intersection returns the best geometry object that corresponds to the set theoretic intersection of the parameter geometry object and this geometry object. Since the intersection of two closed sets is closed, the set theoretic answer should be equal to the topological answer (within the limitation of the metric accuracy of the geometric representations).

#### 6.3.6.6 Set operation: difference (another: GeometryObject): GeometryObject

The operation difference returns the best geometry object that corresponds to the set theoretic difference of the parameter geometry object and this geometry object. Since the difference of two closed sets is not necessary closed, the set theoretic answer may differ from the topological answer along its boundary (within the limitation of the metric accuracy of the geometric representations).

#### 6.3.6.7 Set operation: symDifference (another: GeometryObject): GeometryObject

The operation symmetric difference (symDifference) returns the best geometry object that corresponds to the set theoretic symmetric difference of the parameter geometry object and this geometry object. Since the symmetric difference of two closed sets is not necessary closed, the set theoretic answer may differ from the topological answer along its boundary (within the limitation of the metric accuracy of the geometric representations).

© ISO 2013 – All rights reserved

#### 6.3.6.8 Set operation: union (another: GeometryObject): Geometry

The operation union returns the best geometry object that corresponds to the set theoretic union of the parameter geometry object and this geometry object. Since the union of two closed sets is closed, the set theoretic answer should be equal to the topological answer (within the limitation of the metric accuracy of the geometric representations).

#### 6.3.6.9 Map query operations

##### 6.3.6.9.1 Semantics

The topological query operations are generally dependent of the work following from **Error! Reference source not found.**, which allows them to be completely defined by originally by a 2 by 2 matrix, but later under a 3 by 3 matrix, or possible outcomes of set intersections. This is discussed fully in Clause 12.

##### 6.3.6.9.2 Operation: contains (another: GeometryObject): Boolean

The Boolean operation contains determines whether this object contains the given geometry in 2-dimensions. Contains is a set theoretic contains:

$$A.\text{contains}(B) \Rightarrow A \subseteq B$$

##### 6.3.6.9.3 Operation: crosses (another: GeometryObject): Boolean

The Boolean operation crosses determines whether this object crosses the given geometry in 2-dimensions.

##### 6.3.6.9.4 Operation: disjoint (another: GeometryObject): Boolean

The Boolean operation disjoint determines whether this object is disjoint from the given geometry in 2-dimensions. Disjoint is a set theoretic disjoint:

$$A.\text{disjoint}(B) \Leftrightarrow A \bigcap B = \varnothing$$

##### 6.3.6.9.5 Operation: equals (another: GeometryObject): Boolean

The Boolean operation equals determines whether this object is equals to the given geometry in 2-dimensions. Equals is a set theoretic equals:

$$A.\text{equals}(B) \Leftrightarrow A = B$$

##### 6.3.6.9.6 Operation: intersects (another: GeometryObject): Boolean

The Boolean operation intersects determines whether this object intersects the given geometry in 2-dimensions. Intersects is a set theoretic intersect:

$$A.\text{intersects}(B) \Leftrightarrow A \bigcap B \neq \varnothing$$


© ISO 2013 – All rights reserved

#### 6.3.6.9.7 Operation: overlaps (another: GeometryObject): Boolean

The Boolean operation overlaps determines whether this object overlaps the given geometry in 2-dimensions.

#### 6.3.6.9.8 Operation: touches (another: GeometryObject): Boolean

The Boolean operation touches determines whether this object touches the given geometry in 2-dimensions.

#### 6.3.6.9.9 Operation: within (another: GeometryObject): Boolean

The Boolean operation within determines whether this object is within the given geometry in 2-dimensions. "Within" is the is the reverse of contains:

$$A.within(\mathrm{B}) \Leftrightarrow \mathrm{B.contains(A)}$$

#### 6.3.6.9.10 Operation: withinDistance (another: GeometryObject, distance: Distance): Boolean

The Boolean operation withinDistance determines whether the distance from this object to the given geometry in 2-dimensions is less than the given distance.

#### 6.3.6.9.11 Operation: relate (another: GeometryObject, matrix: CharacterString): Boolean

The Boolean relate operator execute the topological operator represented by the matrix (represented as a row-major string).

### 6.3.7 Interface: Query3D

#### 6.3.7.1 Semantics

The Query3D sub-interface of Geometry Object supplies interfaces for query in 3D coordinate spaces. Since these operations are only distinct from those in Query2D if the objects are 3D spatial, all object supporting Query3D are 3D.

*For instances of Query3D, the derived attribute "is3D" shall always be TRUE.*

#### 6.3.7.2 Attribute: 3Dboundary: GeometryObject

The attribute 3Dboundary generates a 3D geometry object representing the 3D boundary of this object.

#### 6.3.7.3 Attribute: convexHull3D: GeometryObject

The attribute 3Dboundary generates a 3D geometry object representing the 3D convex hull of this object.

#### 6.3.7.4 Operation: 3Ddistance (another: GeometryObject): Distance

The operations 3Ddistance returns the distance between the input geometric object and this geometric object. This distance is the minimal distance between two points in the two objects. If the objects overlap, then the distance is zero.

© ISO 2013 – All rights reserved

This distance is defined to be the greatest lower bound of the set of distances between all pairs of points that include one each from each of the two geometry objects. If necessary, the second geometric object shall be transformed into the same coordinate system as the first before the distance is calculated.

```
3Ddistance (geometry:GeometryObject):Distance
```

The return value may be in one of the units of measure data types defined in ISO TS 19103.

NOTE The role of the reference system in distance calculations is important. Generally, there are at least three types of distances that may be defined between points (and therefore between geometric objects): map distance, geodesic distance, and terrain distance.

Map distance is the distance between the points as defined by their positions in a coordinate projection (such as on a map when scale is taken into account). Map distance is usually accurate for small areas where scale functions have well-behaved derivatives.

Geodesic distance is the length of the shortest curve between those two points along the surface of the Earth model being used by the coordinate reference system. Geodesic distance behaves well for wide areas of coverage, and takes the earth's curvature into account. It is especially handy for air and sea navigation, although care should be taken to distinguish between rhumb line (curves of constant bearing) and geodesic curve distance.

Terrain distance takes into account the local vertical displacements (hypsography). Terrain distance can be based either on a geodesic distance or a map distance.

#### 6.3.7.5 Operation: 3Dbuffer (distance: Distance): GeometryObject

The buffer operation creates a new geometry object that contains all direct positions that are within a distance (less than or equal to the distance) of this geometry object.

Note It is necessary to assure that geometry objects are topologically closed (by definition). Since the distinction between less than a distance or equal to a distance is beyond the capability of any digital computation, this should not matter since the inherent limitation of digital computations makes distinguishing between less than or equal to a distance.

The distance operation returns the distance between the parameter geometry object and this geometry object. The distance will be consistent with the figure of the earth in use, and the distance on the FoE as is consistent with the 3D nature of this interface (Query3D). Thus, the operations buffer creates a new geometric object containing all the direct positions whose distance from this geometric object is less than the radius passed. If the radius is "0", the resulting object is equal to this geometric object. If the radius is negative, the returned geometric object contains all directions positions within this object whose distance from the boundary is greater than the absolute value of the passed radius. The GeometryObject returned is in the same reference system as this original GeometryObject. The topological dimension of the returned GeometryObject is normally the same as the spatial dimension of the reference system - a collection of Surfaces in 2D space and a collection of Solids in 3D space, but this may be application defined.

```
3Dbuffer (radius:Real):GeometryObject
```

NOTE There are cases for which this GeometryObject would be partially outside the domain of validity of the object's reference system. If this case should arise the implementation shall decide on appropriate action.

 © ISO 2013 – All rights reserved

#### 6.3.7.6 Set operation: 3Dintersection (another: GeometryObject): GeometryObject

The operation intersection returns the best geometry object that corresponds to the set theoretic intersection of the parameter geometry object and this geometry object. Since the intersection of two closed sets is closed, the set theoretic answer should be equal to the topological answer (within the limitation of the metric accuracy of the geometric representations).

#### 6.3.7.7 Set operation: 3Ddifference (another: GeometryObject): GeometryObject

The operation difference returns the best geometry object that corresponds to the set theoretic difference of the parameter geometry object and this geometry object. Since the difference of two closed sets is not necessary closed, the set theoretic answer may differ from the topological answer along its boundary (within the limitation of the metric accuracy of the geometric representations).

#### 6.3.7.8 Set operation: 3DsymDifference (another: GeometryObject): GeometryObject

The operation symmetric difference (3DsymDifference) returns the best geometry object that corresponds to the set theoretic symmetric difference of the parameter geometry object and this geometry object. Since the symmetric difference of two closed sets is not necessary closed, the set theoretic answer may differ from the topological answer along its boundary (within the limitation of the metric accuracy of the geometric representations).

#### 6.3.7.9 Set operation: 3Dunion (another: GeometryObject): Geometry

The operation union returns the best geometry object that corresponds to the set theoretic union of the parameter geometry object and this geometry object. Since the union of two closed sets is closed, the set theoretic answer should be equal to the topological answer (within the limitation of the metric accuracy of the geometric representations).

#### 6.3.7.10 Query operations in 3D

##### 6.3.7.10.1 Semantics

The topological query operations are generally dependent of the work following from **Error! Reference source not found.**, which allows them to be completely defined by originally by a 2 by 2 matrix, but later under a 3 by 3 matrix, or possible outcomes of set intersections. The distinction between topological dimensions of the objects does not change the logic of the Egenhofer or Clementini style operations from **Error! Reference source not found.**, **Error! Reference source not found.**, **Error! Reference source not found.**, or **Error! Reference source not found.**. This is discussed fully in Clause 12, and formal definitions of the operators will be given and discussed there.

##### 6.3.7.10.2 Operation: 3Dcontains (another: GeometryObject): Boolean

The Boolean operation contains determines whether this object contains the given geometry in 3-dimensions. Contains is a set theoretic contains:

$$A.3\text{Dcontains}(B) \Rightarrow A \subseteq B$$

© ISO 2013 – All rights reserved

#### 6.3.7.10.3 Operation: 3Dcrosses (another: GeometryObject): Boolean

The Boolean operation crosses determines whether this object crosses the given geometry in 3-dimensions.

#### 6.3.7.10.4 Operation: 3Ddisjoint (another: GeometryObject): Boolean

The Boolean operation disjoint determines whether this object is disjoint from the given geometry in 3-dimensions. . Disjoint is a set theoretic disjoint:

$$A.3\text{Ddisjoint}(B) \Leftrightarrow A \bigcap B = \varnothing$$

#### 6.3.7.10.5 Operation: 3Dequals (another: GeometryObject): Boolean

The Boolean operation equals determines whether this object is equals to the given geometry in 3-dimensions.

#### 6.3.7.10.6 Operation: 3Dintersects (another: GeometryObject): Boolean

The Boolean operation intersects determines whether this object intersects the given geometry in 3-dimensions.

#### 6.3.7.10.7 Operation: 3Doverlaps (another: GeometryObject): Boolean

The Boolean operation overlaps determines whether this object overlaps the given geometry in 3-dimensions.

#### 6.3.7.10.8 Operation: 3Dtouches (another: GeometryObject): Boolean

The Boolean operation touches determines whether this object touches the given geometry in 3-dimensions.

#### 6.3.7.10.9 Operation: 3Dwithin (another: GeometryObject): Boolean

The Boolean operation within determines whether this object is within the given geometry in 3-dimensions. “Within” is the is the reverse of contains:

$$A.3\text{D}\mathit{within}(\text{B}) \Leftrightarrow \text{B}.3\text{Dcontains}(\text{A})$$

#### 6.3.7.10.10 Operation: 3DwithinDistance (another: GeometryObject, distance: Distance): Boolean

The Boolean operation withinDistance determines whether the distance from this object to the given geometry in 3-dimensions is less than the given distance.

#### 6.3.7.10.11 Operation: 3Drelate (another: GeometryObject, matrix: CharacterString): Boolean

The Boolean relate operator execute the above 3D topological operator represented by the matrix (represented as a row-major string).


© ISO 2013 – All rights reserved

### 6.3.8 Interface Empty

The empty set is unique, but it may be represented by any subtype of GeometryObject through the use of the "isEmpty" Boolean attribute (6.3.2.7). There is only one empty set, because the set theoretic definition of equality implies that any ∅ is always equal to any other ∅.

$$A, B \in Sets : [A = B] \Leftrightarrow \left[ [x \in A] \Leftrightarrow [x \in B] \right]$$

It is required in this international standard to allow the set operations (see Clause 6.3.2.24) to have a proper return type where empty results are possible (e.g. the intersection of two disjoint geometries or the difference of any geometry and itself). Since the Empty set does not have a defined dimension, no defined coordinate system, and nothing that distinguishes it from any other ∅, it does not properly belong to any of the primitive classes, but can use the Boolean to be represented by instances of any geometry primitive class. .

Math Some of the behavior of the empty geometry (the empty set, ∅), especially when it interact with itself geometrically, is possibly counter-intuitive for non-mathematicians, so the following facts are given (all mathematically provable from the definitions):

- The distance from the empty set, ∅, to any other non-empty geometry is +∞. This is from the use of glb (greatest lower bound) or min in the definition of distance:

$$distance(A,B) = \min\{c.length | c \in Curve, c.startPoint \in A, c.endPoint \in B\}$$

.

- If either A or B is ∅, then this is the $min(\varnothing) = \text{glb}(\varnothing) = +\infty$ by the definition of the operator "minimum."
- Any buffer, convex hull, boundary, canonical representation or envelop of ∅ is ∅.
- Any "usual" dimension fails but the dim(∅) is defined to be -1 (minus 1) for consistency.
- The Booleans isEmpty, isCycle, isSimple, and isValid are True for ∅.
- The centroid, representativePoint are ∅.
- The return values of stroke, transform, mapProject (or any projection) of ∅ is ∅.
- Any intersection with ∅ is ∅. Any union of ∅ and A is A.
- The difference and symmetric difference of A and ∅ is A.
- The difference of ∅ and anything is ∅.
- The distance from ∅ to ∅ or any other geometry is +∞.
- ∅ is a subset of every set.
- ∅ is a superset only of itself.
- The set ∅ and the set "{∅}" (the set of sets containing only the ∅) are not the same. The cardinality of ∅ is 0. The cardinality of {∅} is 1.

### 6.3.9 Interface GeometryPrimitive

For the purposes of this international standard, a geometry primitive is a connected geometry object with a uniform dimension at every interior point. Depending on the spatial dimension of the coordinate space, the primitives consist of subclasses of Point, Curve, Surface and Solid

-1 Empty set, containing nothing
0 Point, 0-dimensional "positions"; each point is isolated.
1 Curves, 1-dimensional geometry objects where each non-boundary point is

© ISO 2013 – All rights reserved

embedded in a local neighborhood topologically isomorphic to an open interval on the "number line"

2 Surfaces, 2-dimensional geometry objects where each non-boundary point is embedded in a local neighborhood topologically isomorphic to the interior of the unit circle in the Cartesian 2D space, i.e. a plane.

3 Solids, 3-dimensional geometry objects where each non-boundary point is embedded in a local neighborhood topologically isomorphic to the interior of the unit sphere in a Cartesian 3D space.

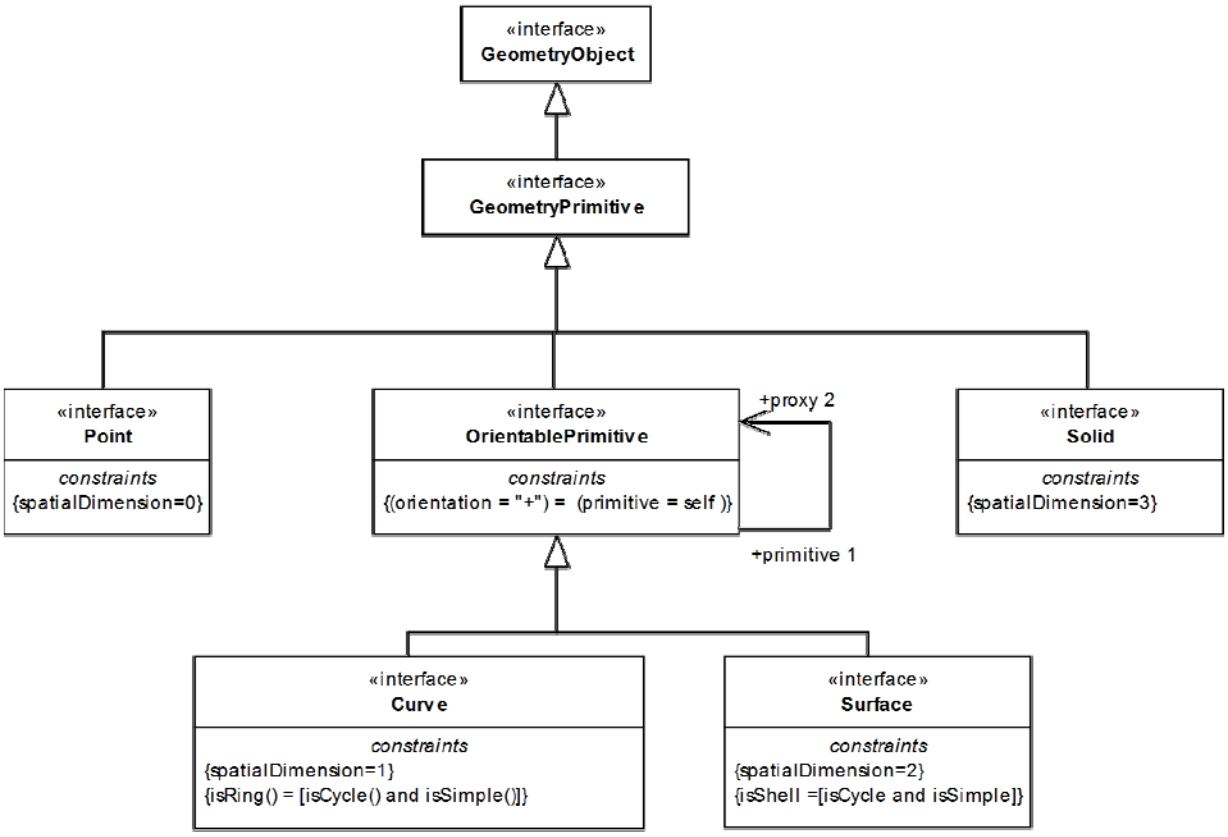



**Figure 12 — Geometry Primitives**

A large number of the geometric types in this standard are defined parametrically, that is they are represented by functions from a set of parameters (in a parametric space, usually a subset of some Euclidean n-dimensional coordinate space or $E^n$) into a coordinate space of some larger dimension. The first few dimensions (up to 3) representing geographic space, the next possibly time, and any remainder representing whatever the application needs, such as distributed attributes or some other measures. The type of geometry is usually determined by the dimension of the parameter space, which will normally be equal to the topological dimension of the resulting geometry. So a 0-parameter geometric object is a point, 1-parameter geometric object is a curve, a 2-parameter geometric object is a surface, a 3-parameter geometric object is a solid. To maintain this, a parametric mapping defining a geometric object should be locally bicontinuous to maintain topological dimension. This means in the parameter space, around any parametric point is a possibly small neighborhood in which the spatial projection of the parametric mapping is one to one with a continuous inverse. This would mean that locally, the geometric object would be an image of n-dimensional Euclidean space, which forces its topological dimension to be the same as the number of parameters.


© ISO 2013 – All rights reserved

In each part of the included model, this version of this international standard includes generalizations of the standard geometric objects (points, curves, surfaces, and solids) to more complex and flexible interpolation algorithms.

The Geometric primitive package contains all the root geometric primitives. Primitives are considered to be the minimal pieces of geometry to be dealt within the context of values. Since most geometric sets are infinite (except for points), primitives can almost always be split into smaller pieces which would also represent geometric primitives, so the usual definition of primitives being non-decomposable cannot be used here. In geometry, primitives are objects that are chosen not to be decomposed into finer geometric objects. They can be decomposed into segments or patches (which are of the same dimension) that are used for representational purposes (such as interpolation).

The closest semantic here is classical Euclidean geometry, where lines, circles and other geometric constructs are treated a single entities. Operation may further break them apart as smaller geometric entities, but the originals still maintain their identity, and structure.

Implementations wishing to combine the functions of geometric primitive types and interpolating structures should use the multiple realizations of these sets of interfaces in defining classes within their application or profile schema. Care should be taken since interface realization carries the intent of substitutability.

#### 6.3.9.1 Semantics

GeometryPrimitive (Figure 12) is the abstract root class of the geometric primitives. Its main purpose is to define the basic "boundary" operation that ties the primitives in each dimension together. A geometric primitive is a geometric object that is not decomposed further into other primitives in the system. This includes curves and surfaces, even though they are composed of curve segments and surface patches, respectively. This composition is a strong aggregation: curve segments and surface patches cannot exist outside the context of a primitive.

NOTE Most geometric primitives are decomposable infinitely many times. Adding a centre point to a line may split that line into two separate lines. A new curve drawn across a surface may divide that surface into two parts, each of which is a surface. This is the reason that the normal definition of primitive as "non-decomposable" is not plausible in a geometry model – the only non-decomposable object in geometry is a point.

Any geometric object that is used to describe a feature is a collection of geometric primitives. A collection of geometric primitives may or may not be a geometric complex. Geometric complexes have additional properties such as closure by boundary operations and mutually exclusive component parts.

GeometryPrimitive and GeometryComplex (see Clause 10.2) share most semantics, in the meaning of operations, attributes and associations. There is an exception in that a GeometryPrimitive shall not contain its boundary (except in the trivial case of Point where the boundary is empty), while a GeometryComplex shall contain its boundary in all cases. This means that if an instantiated object implements GeometryObject operations both as GeometryPrimitive and as a GeometryComplex, the semantics of each set theoretic operation is determined by the its name resolution. Specifically, for a particular object such as CompositeCurve, GeometryPrimitive::contains (returns FALSE for end points) is different from GeometryComplex::contains (returns TRUE for end points). Further, if that object is cast as a GeometryPrimitive value and as a GeometryComplex value, then the two values may not be equal as GeometryObjects.

© ISO 2013 – All rights reserved

### 6.3.9.2 GeometryPrimitive (constructor)

Envelope will often be used in query operations, and therefore must have a cast operation that returns a GeometryObject. The constructor at GeometryPrimitive provides this.

```
GeometryPrimitive::GeometryPrimitive(env:Envelope):
     GeometryPrimitive
```

NOTE The actual return of the operation depends upon the dimension of the coordinate reference system and the extent of the envelope. In a 2D system, the primitive returned will be a Surface (if the envelope does not collapse to a point or line). In 3D systems, the usual return is a Solid.

EXAMPLE In the case where the Envelope is totally contained in the domain of validity of its coordinate system, its associated GeometryPrimitive is the convex hull of the various permutations of the coordinates in the corners. For example, suppose that a particular envelope in 2D is defined as:

```
env:Envelope(lowerCorner(x1, y1), upperCorner(x2, y2))
```

Then we can take the various permutations of the coordinate values to create a list of polygon corners:

```
GeometryCollection(Point(x1,y1),Point(x1,y2),Point(x2,y1),Po
int(x2,y2))
```

If we then apply the convex hull function defined at GeometryObject to the multi_point, we get a polygon,

```
GeometryCollection.convexHull()→ polygon:Surface
```

The extent of a polygon in 2D is totally defined by its boundary (internal surface patches are planar and do not need interior control points) which gives us a data type to represent Surface in 2D:

```
polygon.boundary →
   Line(Point(x1,y1),Point(x2,y1),Point(x2,y2),
Point(x1,y2),Point(x1,y1))
```

So that the surface boundary record is (convex sets have no “interior” holes):

```
boundary:GeometryObject[*]=< exterior=ring, interior={ } >
```

See the relevant clauses for the formal definition of each of these types.

### 6.3.9.3 Association: “Interior to”

The “Interior to” association associates GeometryPrimitives which are by definition coincident with one another. This allows applications to override the Set<DirectPosition> interpretation and its associated computational geometry, and declare one GeometryPrimitive to be “interior to” another.

This association should normally be empty when the GeometryPrimitives are within a GeometryComplex, since in that case the boundary information is sufficient for most cases.

```
GeometryPrimitive::coincidentSubelement[0..n]:Reference<GeometryPr
     imitive>
GeometryPrimitive::superElement[0..n]:Reference<GeometryPrimitive>
```


© ISO 2013 – All rights reserved

This association is constrained by the set theory operators and dimension operators defined at GeometryObject.

```
GeometryPrimitive:
superElement-
     >includes(p:GeometryPrimitive)=GeometryObject::contains(p)
dimension()>=coincidentSubelement.dimension()
```

NOTE This association should not be used when the two GeometryPrimitives are not close to one another. The intent is to allow applications to compensate for inherent and unavoidable round off, truncation, and other mathematical problems indigenous to computer calculations.

A common use of this association might be to list any GeometryObjectProxy objects that use this GeometryPrimitives constructive parameterizations to define themselves.

### 6.3.10 Interface Point

#### 6.3.10.1 Semantics

A Point instance of the GeometryObject is a single location (given by a direct position). See Figure 13

#### 6.3.10.2 Attribute: position: DirectPosition

The attribute "position" gives the location of the Point in its reference system. The distinction between Point and DirectPosition is that Point as an object instance has a system supplied identity, but an instance of DirectPosition is a data type whose only identity is its value.

#### 6.3.10.3 Attribute: boundary: GeometryObject = NULL

The attribute "boundary" gives the boundary of the Point as a GeometryObject reference system. Since the boundary of a point is always empty, the boundary object will always have isEmpty=TRUE.

*For all Points P. P.boundary.isEmpty shall always be NULL.*

$$p \in Point \Rightarrow \left[ p.boundary.isEmpty = TRUE \right]$$

This is the reason that dimension of ∅ is considered to be -1. In most cases, where $\partial$ is the boundary operator, the following is true:

$$\dim(A) = \dim(\partial A) + 1$$

If a point is dimension 0, this would imply that the dimension of ∅ would have to be -1.

© ISO 2013 – All rights reserved

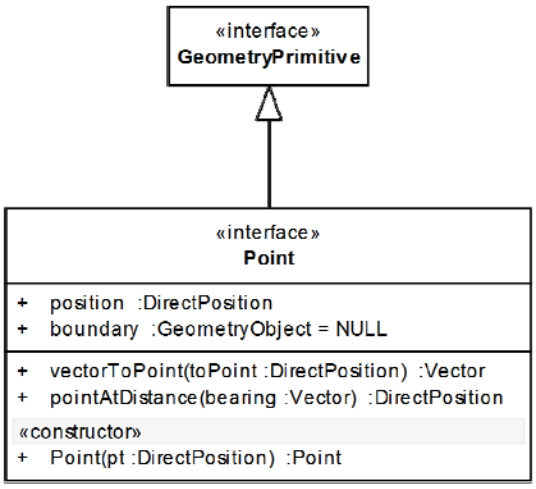



**Figure 13 — Point**

#### 6.3.10.4 Operation: vectorToPoint (toPoint: DirectPosition): Vector

The operation "vectorToPoint" will return a vector in the tangent space at the point whose direction determines a geodesic curve that intersects "toPoint" DirectPosition at a distance equal to the length of the vector. This operation solves the second geodetic problem.

#### 6.3.10.5 Operation: pointAtDistance (bearing: Vector): DirectPosition

The operation "pointAtDistance" will return a DirectPosition given a vector in the tangent space at the point whose direction determines a geodesic curve that intersects that DirectPosition at a distance equal to the length of the vector. This operation solves the first geodetic problem.

#### 6.3.10.6 Constructor: Point (pt: DirectPosition): Point

The constructor Point will create an instance of a Point geometry object, at the given direct Position.

### 6.3.11 Interface OrientablePrimitive

#### 6.3.11.1 Semantics

The interface OrientablePrimitive supplies functionality for primitives that can be meaningfully reversed without changing their "set theoretic" definition.

Orientable primitives (Figure 14) are those that can be mirrored into new geometric objects in terms of their coordinate systems. For curves, the orientation reflects the direction in which the curve is traversed, that is, the sense of its parameterization. When used as boundary curves, the surface being bounded is to the "left" of the oriented curve. For surfaces, the orientation reflects from which direction the local coordinate system can be viewed as right handed, the "top" or the surface being the direction of a completing z-axis that would form a right-handed system. When used as a boundary surface, the bounded solid is "below" the surface. The orientation of points and solids has no immediate geometric interpretation in 3-dimensional space.

OrientablePrimitive objects are essentially references to geometric primitives that carry an "orientation" reversal flag (either "+" = "same orientation") or "-" = "reversed orientation") that determines whether this primitive agrees or disagrees with the orientation of the referenced object.


© ISO 2013 – All rights reserved

NOTE There are several reasons for subclassing the "positive" primitives under the orientable primitives. First is a matter of the semantics of subclassing. Subclassing is assumed to be a "is type of" hierarchy. In the view used, the "positive" primitive is simply the orientable one with the positive orientation. If the opposite view were taken, and orientable primitives were subclassed under the "positive" primitive, then by subclassing logic, the "negative" primitive would have to hold the same sort of geometric description that the "positive" primitive does. The only viable solution would be to separate "negative" primitives under the geometric root as being some sort of reference to their opposite. This adds a great deal of complexity to the subclassing tree. To minimize the number of objects and to bypass this logical complexity, positively oriented primitives are self-referential (are instances of the corresponding primitive subtype) while negatively oriented primitives are not.

Orientable primitives are often denoted by a sign (for the orientation) and a base geometry (curve or surface). The sign datatype is defined in ISO TS 19103. If "c" is a curve, then "<+, c>" is its positive orientable curve and "<-, c>" is its negative orientable curve. In most cases, leaving out the syntax for record "< , >" does not lead to confusion, so "<+, c>" may be written as "+c" or simply "c", and "<-, c>" as "-c". Curve space arithmetic can be performed if the curves align properly, so that:

```
For c, d:OrientableCurves such that c.endPoint=d.startPoint
then
( c + d )==:Composite=< c, d >
```

### 6.3.11.2 Attribute: orientation: sign

The "orientation" of an orientable primitive determines which of the two possible orientations this object represents.

Each instance of an orientable primitive may proxy for itself and for its reverse. The attribute sign distinguishes these two cases, where "+" implies positive orientation, i.e. a proxy for itself, and "-" implies a proxy for its reverse. In 2D and 3D systems, the primitive can be either a curve or a surface. The reverse of a curve swaps start and end and thus determines the direction of parameterization. In a surface, the orientation is the direction of the top of the surface, and a negative "-" reverses top and bottom, in effect reversing the upward normal of the surface.

```
OrientablePrimitive::orientation:Sign
```

© ISO 2013 – All rights reserved

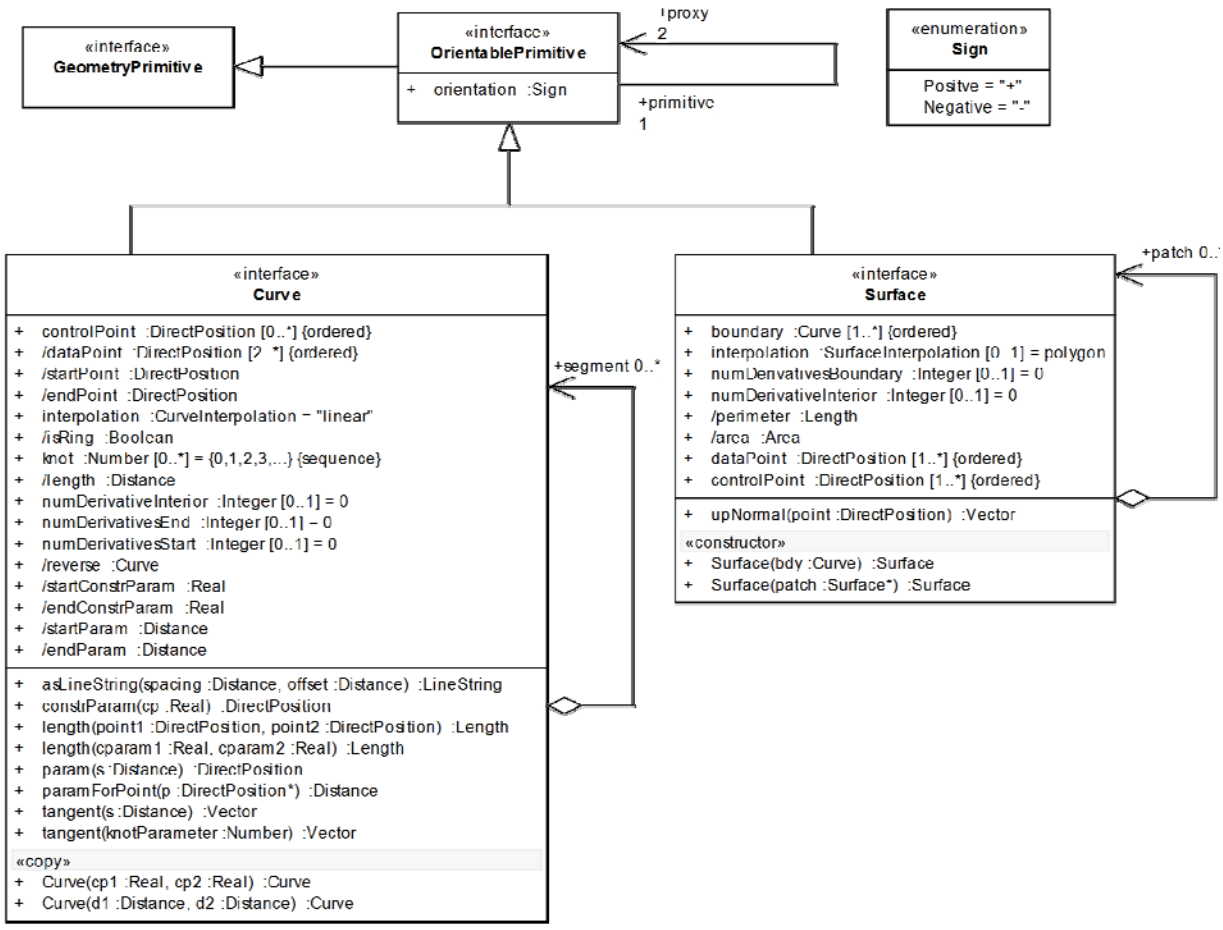



**Figure 14 — Orientable Primitive**

### 6.3.11.3 Association Role: proxy

Each GeometryPrimitive (in pure spatial coordinate system of dimension 1 or 2) is associated to two OrientablePrimitives, one for each possible orientation.

```
GeometryPrimitive::proxy:OrientablePrimitive[0,2]
OrientablePrimitive::primitive:GeometryPrimitive
```

For purely spatial coordinate systems, for curves and surfaces, there are exactly two orientable primitives for each geometric object.

```
OrientablePrimitive:
     (proxy→notEmpty)=(dimension=1 or dimension=2);
     a, b:OrientablePrimitive =>

        ((a.primitive=b.primitive)and(a.orientation=b.orientation
     ))
        implies a=b;
```


© ISO 2013 – All rights reserved

### 6.3.12 Interface: Curve

#### 6.3.12.1 Semantics

The simplest of the orientable primitives is the Curve. This interface contains elements needed for treating curves as geometric objects. In general, it must be kept in mind that there is always a way to choose any point interior to the curve (not the start or end point) and to break the curve at that point to create two curves, whose union is the equivalent geometry of the original curve. This is important in editing procedures where topology or geometric complexes are being maintained.

This international standard does not recognized "degenerate" or "zero-length" curves, which as geometry are really points. Since many exiting implementations do recognize such structures for technical reason usually having to do with system procedures, in such cases care must be taken not to respond to access procedures expecting curves by what are actually points. Since this is an implementation concern, and is unavoidable in some circumstances, conformance to this standard shall require such implementation to supply operations that test whether a "local curve" is actually an instance of Point (is degenerate or zero length) or Curve (is a real geometric curve).

Curve (Figure 15) is a descendent subtype of GeometryPrimitive through OrientablePrimitive. It is the basis for 1-dimensional geometry. A curve is a continuous image of an open interval and so could be written as a parameterized function such as:

$$c(t): (a,\ b) \to \mathbb{R}^n$$

where "$t$" is a real parameter and $\mathbb{R}^n$ is coordinate space of dimension n (as determined by the geometric coordinate system). Curves are continuous, connected, and have a measurable length in terms of the coordinate system. The orientation of the curve is determined by this parameterization, and is consistent with the tangent function, which approximates the derivative function of the parameterization and shall always point in the "forward" direction. The parameterization of the reversal of the curve $c(t):(a,b) \to \mathbb{R}^n$ would have a function of the form:

$$s(t) = c(a+b-t): (a,\ b) \to \mathbb{R}^n$$

Any other parameterization that results in the same image curve, traced in the same direction, such as any linear shifts and positive scales such as

$$e(t) = c\left(a + t\left(b-a\right)\right): (0,1) \to \mathbb{R}^n$$

is an equivalent representation of the same curve. For clarity, this standard recognizes a construction parameter (usually written as '$t$') and an arc length parameter (usually written as '$s$'). The two symbologies $c(t)$ and $c(s)$ represent the same curve, one parameterized by the constructive parameter (from a "knot space") and the other by the arc length of the curve (measured from the start point to the location.

The derivative of a curve $c$, written as $\dot{c}$, is a tangent vector in the tangent space of the FoE at the corresponding point on the curve. If the parameter is arc length, then the tangent is a unit tangent (of length 1):

$$\dot{c}(s) \in T\left(c(s)\right);\ \dot{c}(s) = 1$$

© ISO 2013 – All rights reserved

If the parameter is something other than arc length, then the direction of the tangent is the same but its length is change by the "speed" of the other parameter.

$$\dot{c}(t) \in T(c(t));\ \dot{c}(t) = \left\|\dot{c}(t)\right\| \dot{c}(s) \quad where\ c(t) = c(s)$$

$$\dot{c}(s) = \dot{c}(t) / \left\|\dot{c}(t)\right\| \quad where\ c(t) = c(s)$$

$$\left\|\dot{c}(t)\right\| = \sqrt{(\dot{x}(t))^2 + (\dot{y}(t))^2}$$

If the curve is written in terms of the coordinates of the embedding $\mathbb{E}^3$ space, then if c is written using the knot space:

$$c(t) = (x(t),\ y(t), z(t))$$

Using arc length, the variable "$s$" is used:

$$c(s) = (x(s),\ y(s), z(s))$$

And the tangent vector in the same $\mathbb{E}^3$ space is

$$\dot{c}(t) = (\dot{x}(t),\ \dot{y}(t), \dot{z}(t))$$

And the unit tangent is

$$\dot{c}(s) = (\dot{x}(s),\ \dot{y}(s), \dot{z}(s))$$

In the case where the curve 'c' uses any non-plane Figure of the Earth, the second derivative $\ddot{c}(s)$ is perpendicular to the tangent and can be written as a sum of a vector $\vec{k}_n$ normal to the surface and thus parallel to the surface normal $\vec{N}$, and a vector $\vec{k}_g$ tangent to the surface but normal to the curve's tangent "$\dot{c}$", or using a unit vector $\vec{K}_n$ normal to the surface and a unit vector as $\vec{K}_g$ perpendicular to the tangent:

$$\ddot{c}(s) = \vec{k}_n + \vec{k}_g = \kappa_n \vec{K}_n + \kappa_g \vec{K}_g$$

In these circumstances, the normal curvature vector $\vec{k}_n = \kappa_n \vec{K}_n$ is dependent on the curvature of the surface, and the geodesic curvature vector $\vec{k}_g = \kappa_g \vec{K}_g$ is from the curve itself. If the geodesic curvature is zero ($\kappa_g = 0$) then the underlying curve is a geodesic.

A curve may be 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 possibly the last being the start point of the next segment in the segment list. If the curve is a cycle, then the endpoint of the last is the start point of the first.


© ISO 2013 – All rights reserved

#### 6.3.12.2 Attribute: controlPoint: DirectPosition [0...*]

The controlPoint attribute contains a sequence of coordinate positions to be used in the construction of the curve geometry. The mechanism will depend on the curve type. The control points will have the same initial reference system as the geometry, but may have additional parameters columns for use in the calculation of the data points (see below).

Example In Rational splines, the control points are weighted by a homogeneous coordinate. In circles or circular strings, the additional parameter may indicate the bearing and distance from the control points (center) to the data points.

#### 6.3.12.3 Derived Attribute: dataPoint: DirectPosition [2...*]

The array dataPoint contains a sequence of points on the curve. The first coordinate, dataPoint [0], will be equal to startPoint, and the last, dataPoint[dataPoint.length -1], will be equal to the endPoint. The data points of a curve are in the coordinate space of the geometry.

#### 6.3.12.4 Derived attribute: startPoint: DirectPosition

The startPoint is the coordinate location of the beginning of the curve.

#### 6.3.12.5 Derived attribute: endPoint: DirectPosition

The endPoint is the coordinate location of the terminal end of the curve.

#### 6.3.12.6 Attribute interpolation: CurveInterpolation='linear'

The attribute "interpolation" specifies the curve interpolation mechanism used for this segment. This mechanism uses the control points and control parameters to determine the position of this Curve. The default value is "linear" in the coordinate system.

```
Curve::interpolation:CurveInterpolation="linear"
```

The attribute interpolation is a description of the mechanism using the controlPoint array to calculate the dataPoint array. The default value "linear" implies that the coordinate representation of each segment is a combination of two sequential values in the controlPoint array:

$$\vec{P}(t) = t\vec{P}_n + (1-t)\vec{P}_{n-1}; t \in [0,1]$$

Note If the interpolation is linear, the control points and data points are identical.

#### 6.3.12.7 Derived attribute: isRing: Boolean

The Boolean valued attribute isRing indicates whether the curve is closed and simple, and thus a valid boundary component of a surface.

#### 6.3.12.8 Attribute knot: Number [0...*] = {1, 2, 3...}

Knots are constructive parameterization values that correspond to the dataPoint entries, so that, assuming an array k is the list of knots, and P the list of dataPoint, and c the curve as a function of the construction parameter, then $\boldsymbol{c}(\boldsymbol{k}_n) = \boldsymbol{P}_n$.

© ISO 2013 – All rights reserved

The name "knot" is from spline theory; it means a point in the parameter space of $c(t)$ that is controlled by positions, in the controlPoint and dataPoint arrays. The term knot-space then means the domain from which knots are taken; i.e. the domain of the constructive parameter.

#### 6.3.12.9 Derived attribute length: Distance

The attribute length is the length of the curve. The parameterization of the curve by arc length is thus on the interval $[0, length]$.

#### 6.3.12.10 Attribute numDerivativesInterior: Integer [0...1]

The attribute numDerivativesInterior is the number of continuous derivative guaranteed for values of the construction parameter between the first and the last knot.

#### 6.3.12.11 Attribute numDerivativesStart: Integer [0...1]

The attribute numDerivativesStart is the number of continuous derivative guaranteed for values of the construction parameter just at the startPoint.

#### 6.3.12.12 Attribute numDerivativesEnd: Integer [0...1]

The attribute numDerivativesEnd is the number of continuous derivative guaranteed for values of the construction parameter just at the endPoint.

The attributes “numDerivativesStart” and “numDerivativesEnd” specify the type of continuity between this curve segment and its immediate neighbors, the first value for its predecessor, and the second for its successor. If this is the first or last curve segment in the curve, one of these values, as appropriate, is ignored. The attribute “numDerivativesInterior” specifies the type of continuity that is guaranteed interior to the curve. The default value of “0” means simple continuity, which is a mandatory minimum level of continuity. This level is referred to as “$C^0$” in mathematical texts. A value of 1 means that the function and its first derivative are continuous at the appropriate end point: “$C^1$” continuity. A value of “n” for any integer means the function and its first n derivatives are continuous: “$C^n$” continuity.

```
Curve::numDerivativesAtStart   :Integer[0,1]=0;
Curve::numDerivativesInterior  :Integer[0,1]=0;
Curve::numDerivativesAtEnd     :Integer[0,1]=0;
```

NOTE Use of these values is only appropriate when the basic curve definition is an underdetermined system. For example, line strings and segments cannot support continuity above $C^0$, since there is no spare control parameter to adjust the incoming angle at the end points of the segment. Spline functions on the other hand often have extra degrees of freedom on end segments that allow them to adjust the values of the derivatives to support $C^1$ or higher continuity. Functions to smooth curve segment transitions are important for merging segments while maintaining a level of continuity.

#### 6.3.12.13 Derived attribute reverse: Curve

The reverse of a Curve simply reverses the orientation of the parameterizations of the curve. In most cases this involves a reversal of the ordering of parameters in the curve segments, and a reversal of the order of the segments with a curve.

```
Curve::reverse:Curve
```


© ISO 2013 – All rights reserved

#### 6.3.12.14 Derived attribute startConstrParam, endConstrParam: Real

The attributes startParam and endParam indicate the knot-space parameters for the startPoint and endPoint respectively. The "startConstrParam" and "endConstrParam" indicate the parameters used in the constructive parameterization for the startPoint and endPoint respectively:

```
Curve::startConstrParam:Real
Curve::endConstrParam:Real
Curve:
constrParam(startConstrParam)=startPoint;
constrParam(endConstrParam)=endPoint;
```

There is no assumption that the startConstrParam is less than the endConstrParam, but the parameterization must be strictly monotonic (strictly increasing, or strictly decreasing).

$$c(t):[startConstrParam,\ endConstrParam] \rightarrow \mathbb{R}^n$$

NOTE Constructive parameters are often chosen for convenience of calculation, and seldom have any simple relation to arc distances, which are defined as the default parameterization. Normally, geometric constructions will use constructive parameters, as the programmer deems reasonable, and calculate arc length parameters when queried.

#### 6.3.12.15 Attribute: startParam, endParam: Distance

The attributes startParam and endParam indicate the arc length parameters for the startPoint and endPoint respectively:

```
Curve::startParam:Distance
Curve::endParam:Distance
Curve:
{param(startParam)=startPoint};
{param(endParam)=endPoint};
{length(startPoint, endPoint)=endParam – startParam=length}
```

The start and end parameter of a top level Curve (one which is not in the segment role of another curve) will normally be 0 and the arc length of the curve respectively. For segments within a Curve, the start and end parameters of the segment will normally be equal to those of the Curve where this segment begins and ends respectively in the segment role (see 6.3.12.24), so that the startParam of any segment (except the first) shall be equal to the endParam of the previous segment. If a Curve is used for other purposes, there shall be a restriction that the two parameters must differ by the arc length of the Curve.

$$c(s):[startParam,\ endParam] \rightarrow \mathbb{R}^n$$

#### 6.3.12.16 Operation asLine (spacing: Distance, offset: Distance): Line

The function "asLine" determines an array of Points lying on the original curve that are used to constructs a line string. If "spacing" is given and not zero, then the distance between the points in this sequence shall be not more than "spacing". If "offset" is given and not zero, the distance between the positions on the Line and the original curve shall not be greater than the "offset". If both parameters are set, then both criteria shall be met. If the original control points of the Curve lie on the

© ISO 2013 – All rights reserved

curve, then they shall be included in the points in the Line's controlPoint array. If both parameters are zero, then the implementation may use a general "accuracy" limit universal for the data.

```
Curve::asLine(spacing:Distance=0, offset:Distance=0:Line
```

The "spacing" parameter will control an upper limit of the distance on the original curve of any two data points on the line string. A "0" means no limit is given. The "offset" parameter will control the upper limit of the distance of any point on the original curve between two data points that are preserved in the line string, to some point on the original curve between these same two points. Again a 0 limit means no limit.

*Invoking asLine with two 0 limits for spacing and offset, may return any line sting, but should include at least all of the DirectPositions in the dataPoint array of the original curve.*

NOTE This function is useful in creating linear approximations of the curve for simple actions such as display. It is often referred to as a "stroked curve". For this purpose, the "offset" version is useful in maintaining a minimal representation of the curve appropriate for the display device being targeted (offset = pixel radius). This function is also useful in preparing to transform a curve from one coordinate reference system to another by transforming its control points. In this case, the "spacing" version may be more appropriate.

### 6.3.12.17 Operation constrParam (cp: Real): DirectPosition

The operation constrParam maps the "convenience" parameterization in the knot space to positions on the curve. Knot values will produce entries in the dataPoint array. The operation may be an alternate representation of the curve as the continuous image of a real number interval without the restriction that the parameter represents the arc length of the curve, nor restrictions between a Curve and its component Curves. The most common use of this operation is to expose the constructive equations of the underlying curve, especially useful when that curve is used to construct a parametric surface. The default values of the knot space is an array of integer valued, counting from 0, by 1, up to the number of "knots".

```
Curve::constrParam(cp:Real):DirectPosition
```

### 6.3.12.18 Operation length (point1: DirectPosition, point2: DirectPosition): Length[]

The operation length will return the length of the curve between two points on the curve. If the curve is not simple and passes through the positions more than once, the length returned will be all the possible lengths for the curve, in a monotonic increasing order.

### 6.3.12.19 Operation length (cparam1: Real, cparam2: Real): Length

This variant of length uses constructive parameters (from the knot space) to determine the segment of the curve for which a length is to be calculated.

### 6.3.12.20 Operation param (s: Distance): DirectPosition

The operation "param" shall be the parameterized representation of the curve as the continuous image of a real number interval. The operation returns the DirectPosition on the Curve at the distance passed. The parameterization shall be by arc length, i.e. distance along the Curve measured from the start point and added to the start parameter.


© ISO 2013 – All rights reserved

```
Curve::param(s:Distance):DirectPosition
```

#### 6.3.12.21 Operation paramForPoint (p: DirectPosition): Distance

The operation paramForPoint returns a parameter for the "arc length" interpolation of the curve that most closely returns the original point. If the DirectPosition is not on the curve, the nearest point on the curve shall be used.

```
Curve::paramForPoint(p:DirectPosition):Distance[*]
```

If the passed point "$p:DirectPosition$" is on the curve, and the returned distance is "$d:distance$" then:

```
Curve::param(d)==p and Curve::paramForPoint(p) contains d
```

The DirectPosition closest is the actual value for the "p" used, that is, it shall be the point on the Curve closest to the coordinate passed in as "p". The return set will contain only one distance, unless the curve is not simple. If there is more than one DirectPosition on the Curve at the same minimal distance from the passed "p", the return value may be an arbitrary choice of one of the possible answers.

#### 6.3.12.22 Operation tangent (s: Distance): Vector

The operation tangent returns a tangent vector associated to the coordinate system at the point "s" distance down the curve. If the coordinates in the direct positions are $(x_0, x_1, x_2 . x_3 \ldots.)$ then the coordinates for the vector are multiples of $(dx_0, dx_1, dx_2 . dx_3 \ldots.)$ the differential operators for the coordinates. This vector approximates the derivative of the parameterization of the curve. The tangent with respect to arc length will be a unit vector (have length 1.0), which is consistent with the parameterization by arc length.

```
Curve::tangent(s:Distance):Vector
```

Note If the curve $f_i(t)$ is defined by $f_i(0) = (x_0, x_1, x_2, \ldots, x_i, \ldots.)$ is the point on the curve $c(s)$ where all the coordinates except $x_i$ are constant, and $f_i(t) = (x_0, x_1, x_2, \ldots, x_i + t, \ldots.)$ then $dx_i = \dot{f}_i(0)$ is the tangent for the uniformly increasing coordinate $x_i$.

Care should be taken to remember that the curve $f_i(s)$ is generally not a geodesic, nor are the "vector fields" $(dx_0, dx_1, dx_2 . dx_3 \ldots.)$ non-singular. For example, the fields $d\varphi, d\lambda$ (unit tangents for latitude and longitude) have issues at the poles, basically because the arc length function for changing longitudes is essentially undefinable there.

#### 6.3.12.23 Operation tangent (knotParameter: Number): Vector

The overloaded operation tangent which takes a knotParameter (type is Number) and returns a tangent vector associated to the coordinate system at the point whose constructive parameter was given on the curve. If the coordinates in the direct positions are $(x_0, x_1, x_2 . x_3 \ldots.)$ then the coordinates

© ISO 2013 – All rights reserved

for the vector are $(dx_0, dx_1, dx_2.dx_3 \ldots.)$ the differential operators for the coordinates. If $\vec{\tau}$ is this tangent, and the unit tangent is $\vec{\mu}$ then

$$\frac{dc}{dt} = \vec{\tau}$$

$$\frac{dc}{ds} = \vec{\mu}$$

$$\vec{\mu} = \vec{\tau} \Big/ \|\vec{\tau}\| \Rightarrow \|\vec{\tau}\| = \frac{ds}{dt}$$

where $u$ is the knot parameter and $s$ is the arc length parameter.

### 6.3.12.24 Association Role segment

The role "segment" lists the components (contained smaller Curves) of Curve, each of which defines the direct position of points along a portion of the curve. The order of the segments is the order in which they are used to trace the Curve. Since the composition is recursive, the traversal is specified to be a depth first traversal. First the root curve is used, and then each of its segments curves are used in order of the role. The various parameter intervals are shifted as needed to make the totality of the parameter spaces continuous.

```
Curve::segment:Curve[*]
```

For a particular parameter interval, the Curve and segment curve agree as geometries.

```
Curve:
{curve.startParam≤self.startParam};
{curve.endParam≥self.endParam};
{self.startParam<self.endParam};
{s:Distance(startParam≤s≤endParam)
     implies(curve.parameterization(s)=self.parameterization(s))
     };
```

### 6.3.12.25 Derived attribute boundary: Point [*]

The inherited derived attribute "boundary" on Curve returns the geometry of its topological boundary. If startPoint is not equal to endPoint, the boundary is a two point array. If startPoint is equal to endPoint, the boundary is an empty array.

```
Curve::boundary:Point[*]
```

NOTE The above point array will almost always be two distinct positions, but both Curve and Curves can be cycles in themselves. The most likely scenario is that all of the points used will be transients (constructed to support the return value), except for the startPoint and endPoint of the aggregated Curve. These two positions, in the case where the Curve is involved in a GeometryComplex, will be represented as Points in the same GeometryComplex.

In earlier versions of this standard, this operation was defined separately on the subclasses of Curve.


© ISO 2013 – All rights reserved

#### 6.3.12.26 Operation: length (point1: DirectPosition, point2: DirectPosition): Real

The length of a piece of curvilinear geometry shall be a numeric measure of its length in a coordinate reference system. Since length is an accumulation of distance, its return value shall be in a unit of measure appropriate for measuring distances. The operation "length" shall return the distance between the two points along the curve. The default values of the two parameters shall be the start point and the end point, respectively. If either of the points is not on the curve, then it shall be projected to the nearest DirectPosition on the curve before the distance is calculated. If the curve is not simple and passes through either of the two points more than once, the distance shall be the minimal distance between the two points on this Curve.

```
Curve::length(point1:DirectPosition=startPoint,
                  point2:DirectPosition=endPoint):Real
```

The second form of the operation length shall work directly from the constructive parameters, allowing the direct conversion between the variables used in parameterization and constrParam.

```
Curve::length(cparam1:Real=startConstrParam,
                cparam2:Real=endConstrParam):Real
```

Distances between DirectPositions determined by the default parameterization are simply the difference of the parameter. The length function also allows for the conversion of the constructive parameter to the arc length parameter.

```
If p=length(startConstrParam, p2)+ startParam
then param(p)=constrParam(p2)
```

### 6.3.13 Interface OffsetCurve

#### 6.3.13.1 Semantics

An offset curve is a curve at a predictable distance and bearing from a base curve. They can be useful as a simple alternative to constructing curves that are offsets by definition. For example, if a highway's centerline is stored, then offset from that curve at a constant distance is the right-hand and left-hand side of the road.

The offset curve stores the offset and is can be constructed from the data or control points of the original curve. The offset is a bearing and a distance. If the bearing is absolute, the offset is the geographic equivalent of a parallel translation. If the offset is relative, then the base direction is the tangent of the original curve, and the offset is a motion referenced to the base curve's direction. For example, if the distance is 10 meters, and the bearing is 90°, then the offset curve is the right-hand side of the 10 meter buffer of the original centerline (the small loops generated as in the usual buffer algorithm are removed as they are in buffer generation, and each point on the offset curve is exactly the correct distance from the original curve).

© ISO 2013 – All rights reserved

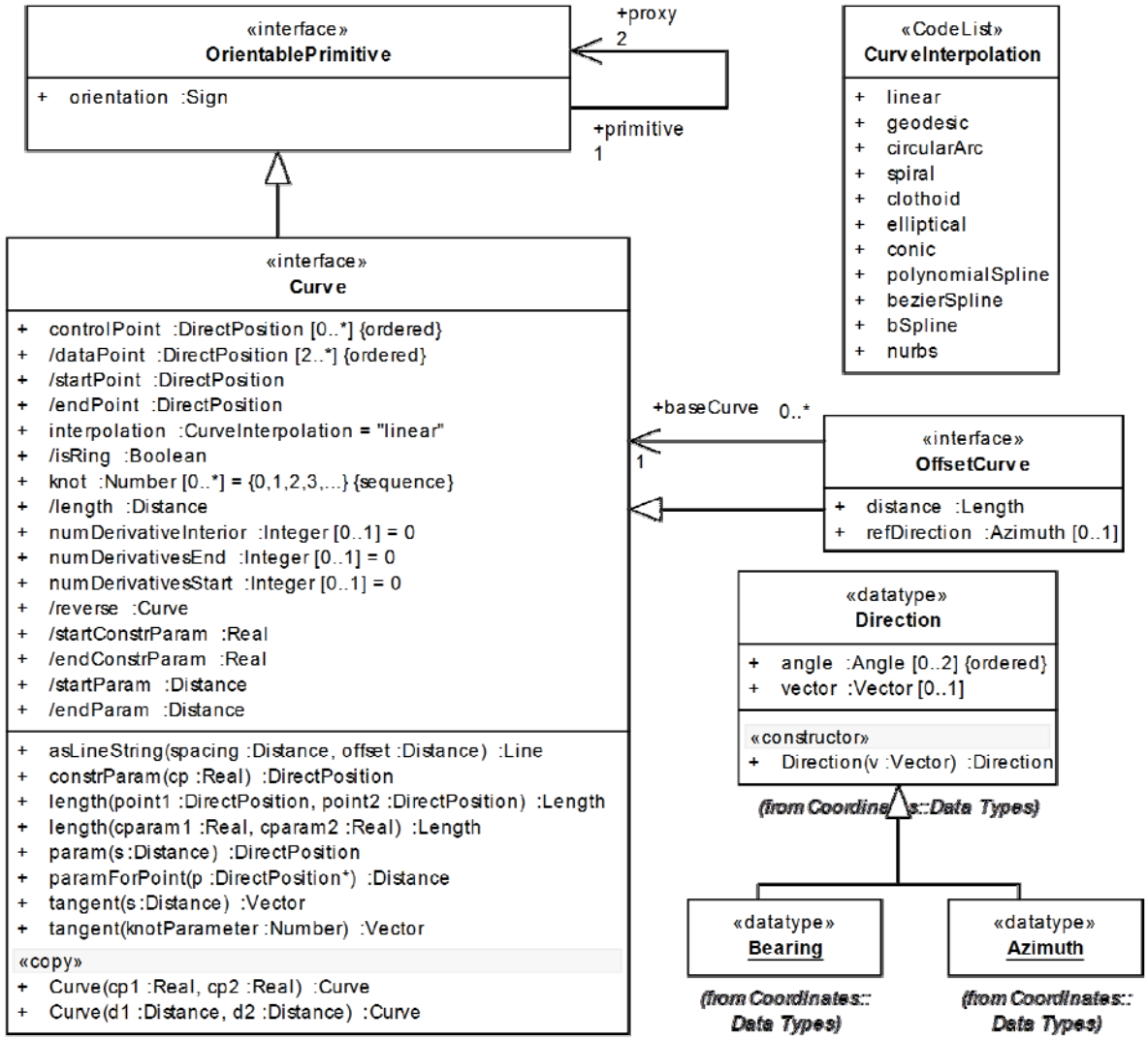



**Figure 15 — Curve and OffsetCurve**

### 6.3.13.2 Attribute: distance

The attribute “distance” is the distance at which the offset curve is generated from the basis curve.

```
OffsetCurve::distance:Distance
```

NOTE This assumes a constant distance from the base curve. Extensions to this standard can address this issue by using a function of position to determine distance (see ISO 19133 and the discussion of linear reference systems, and ISO 19141 and the discussion on moving objects). Such approaches may require augmenting the coordinate system with a linear reference to make the offset function easier to define.

There is a general concept that coverage functions that map positions to values can be used to replace attribute values using the range type of the function. Such extensions are consistent with this international standard, but require mechanisms of their own which are currently out of the scope of this document. In this case, the attribute distance could be replaced by a function of position along the base curve. Such an approach might look like the following, where the domain of the returned coverage function contains the base curve:

```
distance:{ Coverage(position :DirectPosition):Real }
```


© ISO 2013 – All rights reserved

#### 6.3.13.3 Attribute: direction: Bearing[0,1]

The attribute "refDirection" is used to define the vector direction of the offset curve from the basis curve. It can be omitted in cases where the spatial dimension is 2. The distance can be positive or negative. In the 2D case, distance defines left side (positive distance) or right side (negative distance) with respect to the tangent to the basis curve.

In 3D the base curve shall be required to have a well-defined tangent direction for every point. If the curve is not differentiable at some points, then the application may use a reasonable approximation to a smoothly varying tangent vector. The offset curve at any point (parameter) on the basis curve "c" is in the direction $\vec{s} = \vec{v} \times \vec{t}$ where $\vec{v} = c.refDirection$ and $\vec{t} = \dot{c} = c.tangent$

For the offset direction to be well-defined, the refDirection $\vec{v}$ shall not at any point of the curve be in the same, or opposite, direction as $\vec{t}$; which usually means that the curve does not have a purely vertical slope at any point.

```
OffsetCurve::refDirection:Vector[0,1]
```

The default value of the refDirection shall be the local coordinate axis vector for elevation, which indicates up for the curve in a geographic sense.

NOTE If the refDirection is the positive tangent to the local elevation axis ("points upward"), then the offset vector points to the left of the curve when viewed from above.

As in the case for distance, extensions of this standard may replace the constant attribute refDirection with a coverage function of position along the base cure. In classical differential geometry, the binormal vector would often be a good choice. The binormal is perpendicular to both the tangent which is determined by the first derivative of the curve and the normal which is determined by the second derivative of the curve. Issues of continuity of the normal at points of inflection can be handled by sign changes to make the binormal always approximate "up." An especially handy alternative for curves that are on surfaces is to use the surface's UpNormal (see 6.3.15.10).

#### 6.3.13.4 Role: base: Curve

The attribute "base" is a reference to the curve from which this curve is defined as an offset. All non-positional attributes, such as coordinate system, for the offset curve must be the same as for the base curve.

```
OffsetCurve::base:Curve
```

### 6.3.14 CodeList: CurveInterpolation

CurveInterpolation is a list of codes that may be used to identify the interpolation mechanisms specified by an application schema. As a code list, there is no intention of limiting the potential values of CurveInterpolation. Subtypes of Curve can be spawned directly through subclassing, or indirectly by specifying an interpolation method and an associated control parameters record to support it. All interpolations used in this standard have well-known algorithms for splitting a curve segment at any point on the curve and maintaining the underlying geometry object (as a set of DirectPositions). This becomes very important in maintaining topology, and should be maintained (at least to a controllable degree of accuracy) by any extension to this standard that adds interpolation types.

Valid meanings for the set of interpolations include, but are not limited, to the following:

© ISO 2013 – All rights reserved

1. Linear (linear) – the interpolation mechanism shall return DirectPositions on a straight line between each consecutive pair of controlPoints.
2. Geodesic (geodesic) – the interpolation mechanism shall return DirectPositions on a geodesic curve between each consecutive pair of controlPoints. A geodesic curve is a curve of shortest length. The geodesic shall be determined in the coordinate reference system of the Curve in which the Curve is used.
3. Circular arc (circular) – the interpolation mechanism shall return DirectPositions on a circular arc passing through an odd number of datapoints. The sequence of control points shall each be a circle centers which passes through an odd number of data points, and overlap with the previous (if not the first arc) and the next (if not the last arc). The radius of the circle is determined by the geodesic distance from the control point to the first of its datapoints. The second data point should be less than 180° of an arc from the first (which disambiguates the direction of the arc) and the other can be at any further offset in that same direction of rotation. Circles are generated using the geodesic distance. A circle in the tangent plane at a point on a FoE surface, of the given radius can mapped onto a circle on this surface by the exponential map.
4. Spiral — one of the classical spirals is generated at each control point in such a manner so that they form a continuous curve. The spiral type, beginning and ending points and parameters is specified for each control point; the spiral is generated in the tangent space at the control point and mapped onto the surface by the exponential map just as was done with circle.
5. Clothoid (clothoid) – uses a Cornu's spiral or clothoid interpolation (specialization of spiral).
6. Elliptical arc (elliptical) – are generated in a manner similar to circles in the tangent space at the ellipses center and projected on the FoE surface using the exponential map.
7. Conic arc (conic) – same as elliptical arc but using five consecutive datapoints in the tangent space to determine a conic section.
8. Polynomial (polynomial) – the controlPoints are ordered as in a line-string, but they are spanned by a polynomial function. Normally, the degree of continuity is determined by the degree of the polynomials chosen.
9. Bezier Spline (bezier) – the controlPoints are ordered as in a line string, but they are spanned by a polynomial or rational (quotient of polynomials) spline function defined using the Bézier basis. The use of a rational function is determined by the Boolean flag "isRational." If isRational is TRUE then all the DirectPositions associated to the control points are in homogeneous form. Normally, the degree of continuity is determined by the degree of the polynomials chosen
10. B-spline (b-spline) – the controlPoints are ordered as in a line string, but they are spanned by a polynomial or rational (quotient of polynomials) spline function defined using the B-spline basis functions (which are piecewise polynomials). The use of a rational function is determined by the Boolean flag "isRational." If isRational is TRUE then all the DirectPositions associated to the control points are in homogeneous form. Normally, the degree of continuity is determined by the degree of the polynomials chosen.

This list shall be implemented by a code list, and may vary in actual values from the above strings.


© ISO 2013 – All rights reserved

```
CurveInterpolation::
      linear
      geodesic
      circular
      elliptical
      clothoid
      conic
      polynomial
      bezier
      b-spline
      string
      Line
      arcString
```

### 6.3.15 Interface: Surface

#### 6.3.15.1 Semantics

Surfaces are the elements of geometry of 2 topological dimensions.

*A surface shall be connected (contiguous) in such a manner that for any two points on the surface, but not on its boundary, it is possible to create a curve from one to the other in such a manner that the curve is also completely contained in the interior of the surface.*

*A surface shall have as its boundary a set of simple closed curves (call rings). The surface is to the left of all of its rings. These rings shall not self-intersect or be self-tangent.*

They may for rings in the boundary of the same surface, be tangent to one another at a single point, left side to left side. As a set and within the limits of calculation error on the machine in question, the boundary of a surface is uniquely representable by rings. A surface in this specification shall always bounded in the sense that it have a proper envelop, i.e. not include "points at infinity."

Surfaces in 2D will always have a planar interpolation because of the restriction of the dimension of the coordinate space. In 2D because of the restriction above, a surface will have a unique exterior ring, the one with the largest envelope, and some number of interior rings. Each such ring in 2D satisfies the Jordan Curve Theorem and thus divides the space into exactly two regions, one bounded and one unbounded. For rings that are counterclockwise, the bounded area is to the left of the ring as a curve. For rings that are clockwise, the bounded area is to its right. Each ring is said to define an area, the one to its left. A surface in 2D is the set intersection of the areas defined by its rings.

In all cases, the rings in a boundary of a surface shall satisfy the following criteria:

1) They will all be simple, with no self intersections and no self tangencies.
2) They will all be closed, with start point equal to end point.
3) They may be locally tangent to one another, but only once for each pair, only at a point, and only left side to left side.

The instances of Surface, SurfacePatch and any other subtype of GenericSurface shall all satisfy these semantic restrictions.

© ISO 2013 – All rights reserved

Surface (Figure 16) a subclass of GeometryPrimitive and is the basis for 2-dimensional geometry. Unorientable surfaces such as the Möbius band are not allowed as surfaces, but may be constructed as a type of GeometryComplex. The orientation of a surface chooses an "up" direction through the choice of the upward normal, which, if the surface is not a cycle, is the side of the surface from which the exterior boundary appears counterclockwise. Reversal of the surface orientation reverses the curve orientation of each boundary component, and interchanges the conceptual "up" and "down" direction of the surface. If the surface is the boundary of a solid, the "up" direction is "outward." For closed surfaces, which have no boundary, the up direction is that of the surface patches, which must be consistent with one another. The Surfaces instance's included SurfacePatches describe the interior and interpolative structure of a Surface.

NOTE Other than the restriction on orientability, no other "validity" condition is required for Surface.

### 6.3.15.2 Attribute: boundary: Curve []

The attribute boundary will contain Curves which have the Surface on their left hand side. The first Curve may be considered at the exterior boundary, but with the exception of the plane, the concept of a unbounded exterior does not apply.

```
Surface:: boundary: Curve[1…*]
```

In many cases, the boundary curves are derived from the surface, but they may be specified independently of the surface and then used in its construction. For example, the boundary cures in a 2D coordinate system are sufficient to completely determine the "polygon" surface they bound.

### 6.3.15.3 Attribute: interpolation: SurfaceInterpolation

The attribute "interpolation" determines the surface interpolation mechanism used for this SurfacePatch. This mechanism uses the control points and control parameters defined in the various subclasses to determine the position of this SurfacePatch.

```
SurfacePatch::Interpolation:SurfaceInterpolation="polygon"
```

### 6.3.15.4 Attribute: numDerivativesOnBoundary: Integer [0,1]=0

The optional attribute "numDerivativesOnBoundary" specifies the type of continuity between this surface and its immediate neighbors with which it shares a boundary curve. The default value of "0" means simple continuity, which is a mandatory minimum level of continuity. This level is referred to as "$C^0$" in mathematical texts. A value of one means that the functions are continuous and differentiable at the appropriate end point: "$C^1$" continuity. A value of "*n*" for any integer means n-times differentiable: "$C^n$" continuity.

```
SurfacePatch:: numDerivativesOnBoundary :Integer[0,1]
```

NOTE As with curves, any surface and therefore any surface patch must be splitable along any curve on the surface which cuts the surface into at least two distinct pieces. This is not always easily done, and GeometryObjectProxy use may be required. For this reason, any surface patch should have a parameterization from some subset of 2D Euclidean space. This will be discussed at length in the clauses on the various subtypes of surface patch.


© ISO 2013 – All rights reserved

#### 6.3.15.5 Attribute: numDerivativesInterior: Integer [0,1]=0

The optional attribute numDerivativeInterior is the minimal level of continuity within the surface's interior.

#### 6.3.15.6 Derived attribute: perimeter: Length

The derived attribute perimeter will be the sum of the lengths of all boundary curves.

#### 6.3.15.7 Derived attribute: area: Area

The derived attribute area is the total area of the surface

#### 6.3.15.8 Attribute: dataPoint: DirectPosition [0...*]

The potentially derived attribute dataPoint is a collection of points on the surface.

#### 6.3.15.9 Attribute: controlPoint: DirectPosition [0...*]

The attribute controlPoint is a collection of points that are used in the construction of the surface, based on the interpolation type.

#### 6.3.15.10 Operation: upNormal (point: DirectPosition): Vector

The operation upNormal will return a vector perpendicular to the surface at the point. The Normals on the surface will be consistent (the surface will be two-sided). In the case where the coordinate system is 2D, the Normal will be in the 3D embedding Cartesian coordinate space of the Figure of the Earth. In a 3D coordinate system, the upNormal is a vector at the point perpendicular to the tangent plane of the surface at the point.

Note This requires that the surface are orientable, and therefore not some variant of a Mobuis Band or Kline Bottle.

#### 6.3.15.11 Operation: Surface (constructors)

The first version of the constructor for Surface takes a list of surfaces with the appropriate side-to-side relationships and creates a Surface.

```
Surface::Surface(patch[*]: Surface): Surface
```

The second version, which is guaranteed to work always in 2D coordinate spaces, constructs a Surface by indicating its boundary as a collection of Curves organized into a collection of simple, closed curves. In 3D coordinate spaces, this second version of the constructor shall require all of the defining boundary Ring instances to be coplanar (lie in a single plane) which will define the surface interior interpolation method as planar.

```
Surface::Surface(boundary:Curve[*]):Surface
```

© ISO 2013 – All rights reserved

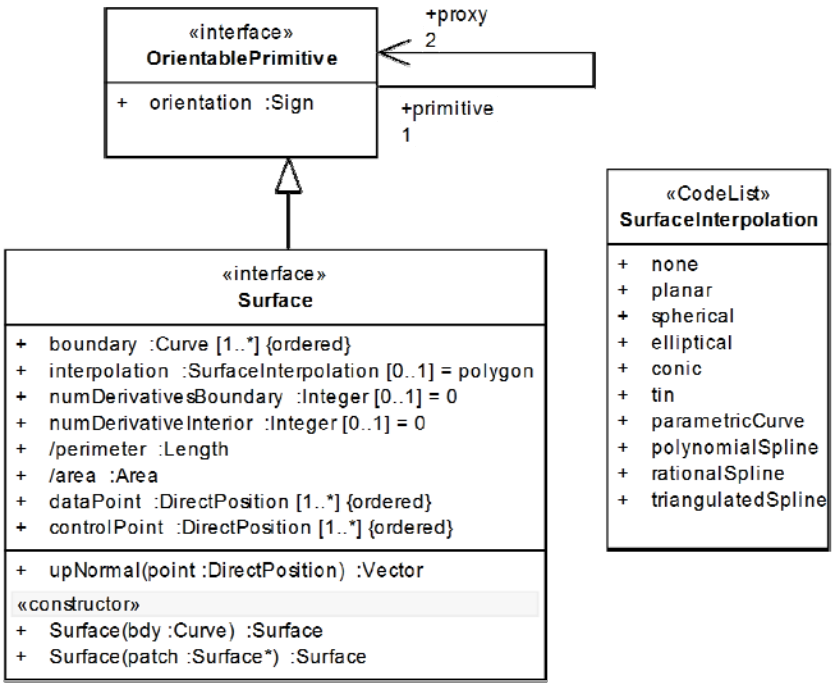



**Figure 16 — Surface**

### 6.3.16 CodeList: SurfaceInterpolation (to be updated upon completion of interpolation clauses)

SurfaceInterpolation (Figure 16) is a list of codes that may be used to identify the interpolation mechanisms specified by an application schema. Valid values for "interpolation" include, but are not limited, to the following:

1. None (none) – the interior of the surface is not specified. The assumption is that the surface follows the reference surface defined by the coordinate reference system.
2. Planar (planar) – the interpolation method shall return points on a single plane. The boundary in this case shall be contained within that plane.
3. Linear (linear or bilinear) – control points (some of which may be outside of the surface boundary) are organized into 2D arrays, and each square in the array uses the unit square in $\mathbb{E}^2$. $\{(x,y)|0 \le x \le 1, 0 \le y \le 1\}$. And then interpolating using bilinear interpolation.
4. Spherical (spherical), Elliptical (elliptical), Conic (conic) – the surface is a section of a spherical, elliptical or conic surface.
5. Triangular (triangle) – the control points are organized into adjoining triangles, which form small planar segments. This means the surface is a triangular irregular network (TIN).
6. Parametric Curve (parametric Curve) – the control points are organized into a 2-dimensional grid and each cell within the grid is spanned by a surface which shall be defined by a family of curves.
7. Polynomial Spline (polynomialSpline) – the control points are organized into an irregular 2-dimensional grid and each cell within this grid is spanned by a piecewise polynomial spline function. In the case of Bézier or B-Splines, the more specific term is used.
8. B-Spline (b-spline) – the control points are organized into an irregular 2-dimensional grid and each cell within this grid is spanned by a basis spline function. If the coordinates are in homogeneous format, this is a rational spline, and the associated Boolean flag "isRational" will be set to TRUE,
9. Bezier Spline (bezier) – the control points are organized into irregular 2-dimensional grid, each cell within this grid is spanned by a Bézier spline function. If the coordinates are in


© ISO 2013 – All rights reserved

homogeneous format, this is a rational spline, and the associated Boolean flag “isRational” will be set to TRUE,

If more than one interpolation description fits the method used, then the most restrictive one will be used.

```
SurfaceInterpolation::
none
planar
bilinear
spherical
elliptical
conic
tin
parametricCurve
polynomialSpline
bezier
b-spline
nurbs
```

### 6.3.17 Interface Solid

#### 6.3.17.1 Semantics

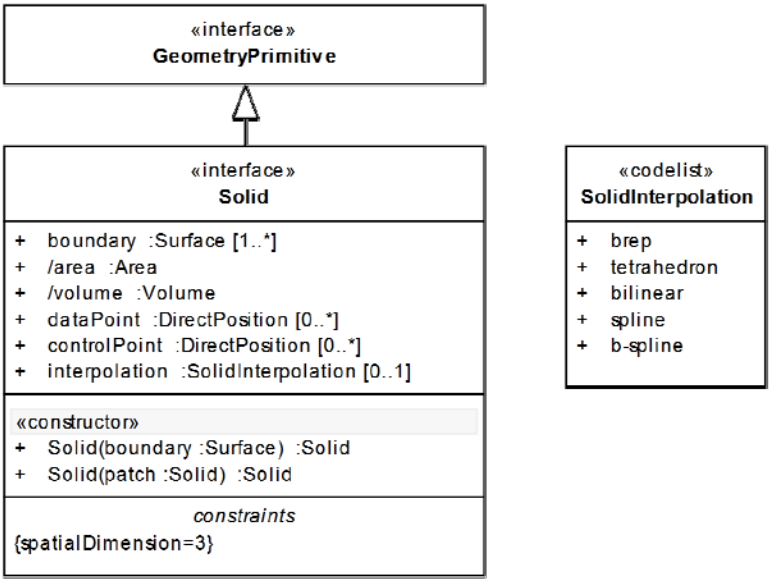



**Figure 17 — Solid**

The Solid primitive contains interfaces needed for treating Solid as geometric objects. Since a solid is by definition, a 3D topological object, it can only exist in a coordinate space with 3 spatial dimensions.

In a 3D world it is sufficient to define surface by their boundary (B-REP or boundary representation). In cases of higher dimensions, such as may be found in coverage functions, some types of solid modeling with deal with non-homogeneous interior structures or in cases where parameterizations are used to map “template geometries” into real space, a more complex but more capable approach may be useful. In these cases, the structure of the interior of the solid may be represented by functions based on the same sort of mathematics used to embed surfaces in 3D space.

© ISO 2013 – All rights reserved

In all cases, a solid has a boundary consisting of surfaces in the same coordinate system as the solid. These surfaces can be organized into shells which are the 3D equivalent to the rings for a surface. A solid shall be connected (contiguous) in such a manner that for any two points on the solid, but not on its boundary, it is possible to create a curve from one to the other in such a manner that the curve is also completely contained in the interior of the solid. A solid shall have as its boundary a set of simple closed surfaces (shells). A solid is always below it shells (as defined by the upward normal of the surface). These shells shall not self-intersect nor be self-tangent. They may, for shells in the boundary of the same solid, be tangent to one another at a single point or along a non-closed, simple curve, bottom side to bottom side (that means that the upward normals of the two at points of contact are in opposite directions). As a set and within the limits of calculation error on the machine in question, the boundary of a solid is uniquely representable by shells (as a collection of surfaces that are both cycles (closed) and simple). A solid in this specification shall always bounded in the sense that it have a proper envelop, i.e. not include "points at infinity."

Solids in 3D will always have a simple interpolation because of the restriction of the dimension of the coordinate space. In 3D because of the restriction above, a solid will have a unique exterior shell, the one with the largest envelope, and some number of interior shells. Each such shell in 3D satisfies the Jordan-Schönflies (Jordan Separation) Theorem and thus divides the space into exactly two regions, one bounded and one unbounded. Each shell is said to define a volume, the one in the direction opposite from the upward normal.

Solid (Figure 17), a subclass of GeometryPrimitive, is the basis for 3-dimensional geometry. The spatial extent of a solid is defined by the boundary surfaces.

#### 6.3.17.2 Attribute: boundary: Surface [1...*]

In instantiations of this interface, which will always be instantiations of GeometryObject, the attribute "boundary" specializes the boundary defined at GeometryObject and at GeometryPrimitive with the appropriate return type. It shall return a set of Surfaces that limit the extent of this solid. These surfaces shall be organized into one set of surfaces for each component of the solid. Each of these shells will be a cycle (closed composite surface without boundary).

```
Solid::boundary():Surface [1…*]
```

NOTE The exterior shell of any single-component solid is defined only because the embedding coordinate space is always a 3D Euclidean one; i.e. the 3D space in which the Figure of the Earth for its coordinates is defined. In general, a solid in a bounded 3-dimensional manifold has no distinguished exterior boundary.

In cases where "exterior" boundary is not well defined, all the shells of the solid boundary shall be listed as "interior". This can only happen in a minimum of 4 spatial dimensions and is beyond the scope of this international standard.

The shells that bound a solid shall be oriented outward – that is, the "top" of each Shell as defined by its "upward normal" orientation shall face away from the interior of the solid.

#### 6.3.17.3 Derived attribute: area: Area

The derived attribute "area" shall specify the sum of the surface areas of all of the boundary components of a solid.

```
Solid::area():Area
```


© ISO 2013 – All rights reserved

The array class Surface [*] has a “column operation” called “area” that accumulates the area of the components of the set. Using this, it can be said that for a Solid:

```
Solid: area()=self.boundary.area()
```

#### 6.3.17.4 Derived attribute: volume: Volume

The derived attribute “volume” shall specify the volume of this solid. This is the volume interior to the exterior boundary shell minus the sum of the volumes interior to any interior boundary shell.

```
Solid::volume: Volume
```

#### 6.3.17.5 Attribute dataPoint: DirectPosition [0...*]

The optional dataPoint array can carry point sample information about the interior and boundary of the solid. In general, the datapoints can be treated as a “point cloud” of information, but the various subtypes of solid can add requirements on this array. The directPosition values of this array can carry as many “parameter” columns as required by the reference system associated to the object as geometry in Clause 6.3.2.2. The interpolation mechanism can be found in the interpolation attribute in Clause 6.3.17.7.

#### 6.3.17.6 Attribute controlPoint: DirectPosition [0...*]

The optional controlPoint array can often be used similar to the dataPoint array, except controlPoint positions do not need to be contained in the solid. The interpolation values from control points are only valid if they are contained in the boundary of the solid.

#### 6.3.17.7 Attribute: interpolation: SolidInterpolation [0...1]
#### 6.3.17.8 Role: component: Solid

The optional association role “patch” gives a Solid access to these parameterizations of space. If this role is non-empty, the union of the images of the patches should have the same boundary as the associated solid. In all cases the extent of the Solid is defined by its boundary, not by it

```
Solid::patch:SolidPatch[*]
```

#### 6.3.17.9 Solid (constructor)

Since this International Standard is limited to 3-dimensional coordinate reference systems, any solid is definable by its boundary. The default constructor for a Solid is from a properly structured set of Shells organized as a SolidBoundary.

```
Solid::Solid(boundary:SolidBoundary):Solid
```

### 6.3.18 CodeList: SolidInterpolation (to be updated upon completion of interpolation clauses)

SolidInterpolation is a list of codes for 3D objects that may be used to identify the interpolation mechanisms specified by an application schema. Valid values for “interpolation” include, but are not limited, to the following:

© ISO 2013 – All rights reserved

1. None (none) – the solid is solely defined by its boundary
2. Linear (linear or trilinear) – control points (some of which may be outside of the solid boundary) are organized into 3D arrays, and each cube in the array uses the unit cube in $\mathbb{E}^3$. $\{(x,y,z)|0 \le x \le 1, 0 \le y \le 1, 0 \le z \le 1\}$. And then interpolating using trilinear interpolation.
3. Tetrahedron (tetrahedron) – the space will be broken into some number of tetrahedrons, each using a local linear interpolation. Using each tetrahedron is given by 4 control points would use Barycentric coordinates to parameterize the convex hull of these 4 points. Solids using this method will always be convex hulls of their corners. This in essence maps a standard 4-space Euclidean parameter space define by $\{(u_0,u_1,u_2,u_3),|\ 0 \le u_i \le 1, u_0+u_1+u_2+u_3=1\}$ to a standard 3-space Euclidean tetrahedron defined by the points $(0,0,0),(1,0,0),(0,1,0),(0,0,1)$ and thence to the coordinate system (regardless of dimension) of the Solid being defined. This mechanism mimics the triangulated irregular network in 2D.
4. Parametric Curves (parametricCurve) – the control points are organized into irregular 3-dimensional grid, each line parallel to a dimensional axis in the parameter space is associated to a curve parameterization using the appropriate control points, and the union of all the curves images can result in a full 3D parameterization of the space. This is similar to the parametric curve surface approach defined for surface interpolations.
5. Polynomial splines (polynomialSpline) – a parametric curve interpolation where the interpolating curves are spline functions.
6. Bezier Spline (bezier) – the control points are organized into irregular 3-dimensional grid, each cell within this grid is spanned by a Bézier spline function. If the coordinates are in homogeneous format, this is a rational spline, and the associated Boolean flag "isRational" will be set to TRUE.
7. B-Spline (b-spline) – the control points are organized into an irregular 3-dimensional grid and each cell within this grid is spanned by a basis spline function. If the coordinates are in homogeneous format, this is a rational spline, and the associated Boolean flag "isRational" will be set to TRUE.

### 6.3.19 Interface: GeometryCollection

#### 6.3.19.1 Semantics

A geometry collection (see Figure 18) is a type collection of geometry objects which acts as a set union of its elements. The operations add and remove act as union and difference, not necessarily as would be expected as a set of sets. Implementations wishing to implement a "set of sets" cannot use Geometry Collections (a subtype of GeometryObject, but could use a simple array of geometry objects.


© ISO 2013 – All rights reserved

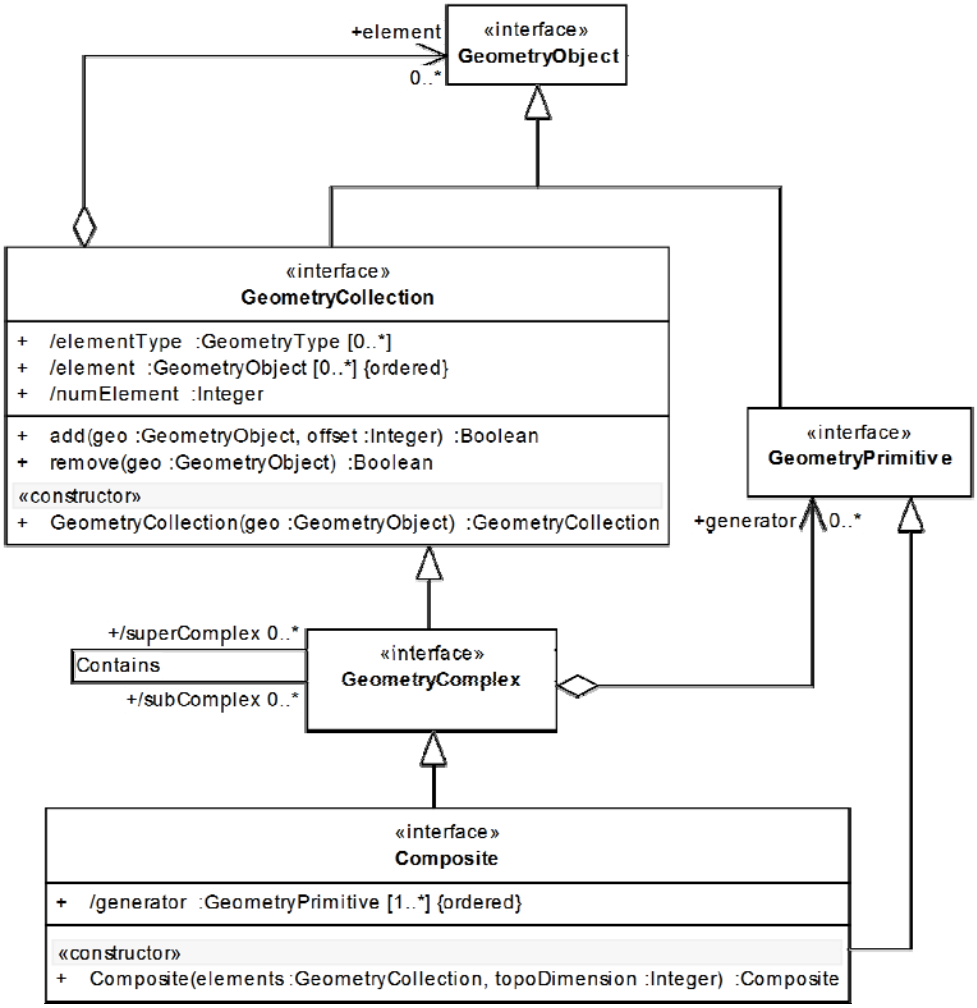



**Figure 18 — Geometry Collection**

### 6.3.19.2 Attribute: GeometryCollection:: elementType: GeometryType [0...*]

The attribute element type is a list of Geometry Types that are allowed in this collection. The value of this attribute is normally set at construction based on the purpose for this particular instance of geometry collection

*Any element in a geometry collection shall be a type listed in the element type array, or be a subtype of a type listed.*

*If the collection is to be used for a particular purpose that would limit the types listed, then the attribute elementType may be read-only.*

### 6.3.19.3 Attribute: GeometryCollection:: element: GeometryObject [0...*]

The array attribute supplies the list of geometry object in this collection.

*Each object in the collection shall be an instantiation of one or more of the types listed in the elementType array.*

© ISO 2013 – All rights reserved

#### 6.3.19.4 Attribute: GeometryCollection:: numElement: Integer

The read-only attribute numElement returns the count of the elements in the collection at the moment of the access to the property. The only manner to change the numElement value is to add or delete elements in the collection.

#### 6.3.19.5 Role: GeometryCollection:: element: GeometryObject

The role element is an alternate access point to the elements in the Geometric object. The two logical arrays attribute: element and role: element are identical.

#### 6.3.19.6 Operation: GeometryCollection:: add (geo: GeometryObject, offset: Integer): Boolean

The operation "add" inserts a new element into the collection. The geometry collection acts like a set of direct positions, and its "canonical representation" may combine many of the elements added to the collection

*The set theoretic effect of add shall be equal to the set union this object and the object passed as geo.*

*If offset is non-empty, then the new object will be listed in the element array (from the relation "element") at the offset position specified.*

#### 6.3.19.7 Operation: GeometryCollection:: remove (geo: GeometryObject): Boolean

The operation "remove" deletes an existing element from the collection.

*The set theoretic effect of a remove shall be equivalent to a symmetric difference of this object and the object passed as geo.*

#### 6.3.19.8 Constructor: GeometryCollection (geo: GeometryObject []): GeometryCollection

The result of the constructor which takes an array of GeometryObjects is to union the input objects as a set of sets of DirectPositions.

#### 6.3.19.9 Association: element: GeometryObject

The "element" association is an ordered array of linkages to GeometryObjects.

*The type of any element member of the association shall be a type listed in the elementType array, or subtype of such a type.*

### 6.3.20 Interface: GeometryComplex

#### 6.3.20.1 Semantics

For the purposes of this international standard, a geometry primitive is a connected geometry object with a uniform dimension at every interior point.


© ISO 2013 – All rights reserved

*A geometry complex shall be connected.*

A geometry complex (see Figure 10) begins with a collection of primitives of the same dimension (called the "generators") with disjoint interiors, i.e. they only intersect on common boundaries. Other primitives are added so that the collection is "complete" under the boundary operator, i.e. the boundary of any primitive in the collection has its boundary representable as other primitives in the GeometryComplex.

*All direct positions contained in a geometry complex shall be interior to one and only one primitive.*

At each dimension, the set of primitives in the complex of that dimension forms the generators of a complex of that dimension. For a 3D complex, the structure works as follows:

- For each dimension, n= 3, 2, 1 and 0, there exists a collection $C_n$ of geometric primitives of that dimension with disjoint interiors such that the elements of the boundary of objects in $C_n$ are in $C_{n-1}$.

$$C_3 \xrightarrow{\partial} C_2 \xrightarrow{\partial} C_1 \xrightarrow{\partial} C_0 \xrightarrow{\partial} \varnothing$$

- G, the geometric complex is the disjoint union of the interiors of the elements of $C_{0,}\ldots$ Cn.
- G is also the union of the generators (as geometry objects, the generators are closed).

### 6.3.20.2 Role: GeometryComplex: generator: GeometryPrimitive []

The array associated to the role generator contains the primitives of the top dimension of the complex ( $C_{\text{topologicalDimension}}$ ).

```
GeometryComplex::generator: GeometryPrimitive [*]
```

### 6.3.20.3 Role: GeometryComplex: superComplex, subComplex: Geometric complex

A superComplex is any complex which contains all the elements of the current complex, and is hence a super set of this object, both as a geometry object and as a collection of geometry objects. A subComplex is contained in the "superComplex" target, and is a subset of it both as a geometry object and as a collection of geometry objects.

### 6.3.20.4 Cellular complexes (informative example for use in topological examples)

There are standard "primitives" called cells for each dimension. In each Euclidean space there are two standard geometries defined, a disk and a sphere:

$D^n = \{p \in \mathbb{E}^n \mid p \le 1\}$ i.e., all points in $\mathbb{E}^n$ in the unit disk centered at the origin

$S^{n-1} = \{p \in \mathbb{E}^n \mid p = 1\}$ i.e., all points in $\mathbb{E}^n$ on the unit sphere centered at the origin

From a topological point of view, the boundary of the disk is the sphere: $\partial D^n = S^{n-1}$; and the disk is the union of its "northern (positive last coordinate)" and "southern (negative last coordinate)"

© ISO 2013 – All rights reserved

hemisphere, each of which is topological equivalent to a disk: $S^n \approx D^n_{south} \bigcap D^n_{north}$ . By slicing the sphere into as many equal dimensioned disks as necessary, any geometric structure can be represented by collections of cells each topologically isomorphic to an appropriately dimension n-disk. Thus, it turns out that all topological structures of general interest to geometry can be represented by a cellular complex in which each primitive is topological equivalent to a disk of the proper dimension.

The purpose of the definition of simple for geometry objects is to keep the decomposition in "cells" easy.

### 6.3.21 Interface: Composite

#### 6.3.21.1 Semantics

Composites will be collections that also implement one of the primitive type, identified by the element s' topological dimension. Any collection of curves which can be combined into a single curve (with potentially multiple interpolations on its subsegments) can be represented by a composite of topological dimension 1. This is similar to composite surfaces (dimension 2) and composite solids (dimension 3).

*A composite with a topological dimension -1 is the empty set ∅ as a geometry instance .*

*A composite with a topological dimension 0 shall also implement Point.*

*A composite with a topological dimension 1 shall also implement Curve.*

*A composite with a topological dimension 2 shall also implement Surface.*

*A composite with a topological dimension 3 shall also implement Solid.*

Note Common usage will refer to a composite of dimension 1 as a "composite curve" or "curve string." A composite curve consisting of lines is a "composite line" or "line string." This extends to other primitives such as "composite arc" or "arc sting" as a "composite arc" or "arc string." It would be possible to create different "special" interfaces for each "sement type, but the behavior is the same for these homogeneous "strings" and for other heterogeneous stings consisting of distinct segments types. Spline curves can change interpolation functions at knots and can therefore often be interpreted as stings.

Similar names for surfaces might be "polyhedral surfaces" (any set of polygons) or triangulated surfaces (for triangles only). Spline surfaces and NURBS are also composites where the interpolation functions vary over the area defined by the knot sets.

#### 6.3.21.2 Inherited attribute: topologicalDimension: Integer

The topologicalDimension constrains the elementType to the subtypes of a geometric primitive type; either point, curve, surface or solid (0, 1, 2, 3 respectively).

*A composite with a topological dimension 0 shall contain elements which are subtypes of Point.*

*A composite with a topological dimension 1 shall contain elements which are subtypes of Curve, ordered and oriented so that each Curve after the first begins with a startPoint equal to the endPoint of the previous Curve. .*

 © ISO 2013 – All rights reserved

*A composite with a topological dimension 2 shall contain elements which are subtypes of Surface, ordered so that each Surface is adjacent the ones before or after it in the list.*

*A composite with a topological dimension 3 shall contain elements which are subtypes of Solid ordered so that each Solid is adjacent the ones before or after it in the list. .*

# 7 Interpolations for Curves

## 7.1 Requirements class: Lines

### 7.1.1 Semantics

This package contains the first curve segments based on the path of shortest distance between points. In a Euclidean space such as a planar FoE, this is the straight lines between points. Some arguments have been made that "line" and "geodesic" should be the same class, but the term "line" is often used to mean a "linear interpolation" between points based on the coordinate system. In most all cases in geography where the "surface is curved" this differs from "geodesic" which is always the part of shortest length between nearby points.

Math In differential geometry, the geodesics (especially on surfaces in 3-space) are usually defined as ones in which the "normal" (the direction of the 2nd derivative of the curve with respect to arc length) is orthogonal to the surface. It is an exercise in the calculus of variations to show that these are the curves of shortest distance constrained by the surface.

### 7.1.2 Interface: Line

#### 7.1.2.1 Semantics

A Line (Figure 19) consists of sequence of line segments, each having a parameterization between two consecutive controlPoints.

The controlPoints (inherited from Curve) of a Line are a sequence of positions between which the curve is linearly interpolated. The controlPoint and dataPoint arrays are identical for line stings. The first position in the sequence is the startPoint of the Line, and the last point in the sequence is the endPoint of the Line.

```
Line::point(≡dataPoint):DirectPosition[2…*]
```

#### 7.1.2.2 Attribute: interpolation: CurveInterpolation ≡ linear

For the class, Line, the interpolation attribute (inherited from curve) is always "linear." The controlPoint and dataPoint arrays are identical, and may reference the same internal storage. Each segment between any two of these point is always a line segment in $\mathbb{R}^n$ (n is the dimension of the direct position in the point array) as a vector space. The effective interpolation between any two points (with index "i" and "i+1" in the point array) is:

$$c(\lambda) = (1-\lambda)\vec{P}_i + \lambda\vec{P}_{i+1} \text{ for } \lambda \in [0,1]$$

The knot array is the sequence of real numbers that mark to movement from one segment to another.

$$c(t) = \left(1 - \frac{\left(t-k(i)\right)}{\left(k(i+1)-k(i)\right)}\right)\vec{P}_i + \left(\frac{\left(t-k(i)\right)}{\left(k(i+1)-k(i)\right)}\right)\vec{P}_{i+1} \text{ for } t \in \left[k(i), k(i+1)\right]$$

© ISO 2013 – All rights reserved

For the entire curve, the first knot is the 'startConstrParam' of the curve, and the last knot used is the 'endConstrParam' of the curve. In the above equation, a local variable $\lambda$ is defined from the constructive parameter $t$ and the values of the consecutive knots in the knot array:

$$\lambda = \frac{\left(t - k(i)\right)}{\left(k(i+1) - k(i)\right)} \text{ for } k(i) \le t \le k(i+1)$$

#### 7.1.2.3 Line (constructor)

The constructor for Line takes a sequence of points and constructs a Line with those points as the controlPoints/dataPoints.

```
Line::Line(point:DirectPosition[2…*]): Line
```

NOTE If all of the DirectPositions passed to this constructor are collinear (all lie on the same line) then the result is essentially a line segment, and can be represented as such.

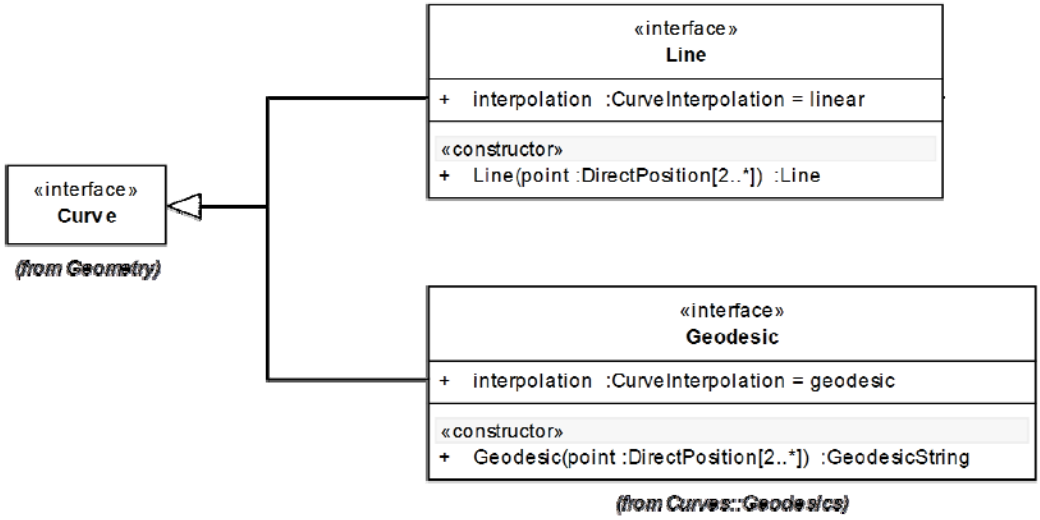



**Figure 19 — Linear and Geodesic curves**

### 7.2 Requirements class: Geodesics

#### 7.2.1 Semantics

In a more general setting, where the Figure of the Earth is not flat but curved (as on the "real" surface of the Earth), the shortest path between points is a geodesic. The term applies on any manifold that has an associated Gaussian metric (which acts as a "dot" product of the vectors in the tangent space at each point), including the situation where geometry gets is name ("geo" means Earth).

Math On a sphere, the planes through the center of the sphere cut the sphere in great circles. The 2nd derivative of a circle is an inward vector pointing towards the center of the circle and constrained by both the plane and the sphere. Therefore, the normal is orthogonal to the sphere and the great circles are geodesics. On a Mercator projection of the sphere the great circles are "sine wave" looking curves that pass through the equator every 180° (the intersection of the equator and the other circles are antipodal – perfect opposite positions on the sphere).

On an ellipsoid, the meridian and equatorial planes cut the surface in such a manner as to be geodesics. The other geodesics (in a Mercator projection of the ellipsoid) are the same sort of "sine wave" looking curves, and they do pass through the equator, but the period of their passing is smaller than the 180° of the sphere. These geodesics are not planar.


© ISO 2013 – All rights reserved

### 7.2.2 Interface: Geodesic

#### 7.2.2.1 Semantics

A Geodesic (Figure 19) consists of sequence of geodesic segments. The class is an array of geodesic segments as a single object.

MATH A geodesic is a curve on the Figure of the Earth being used; since the FoE is embedded in a Euclidean 3-space, $\mathbb{E}^3$, it is also a curve in 3-space. The tangent vector $\dot{c}(t)$ at a point on the surface is also a vector in $\mathbb{E}^3$. If it is normalize (made a unit vector) the result is called the "unit speed" tangent ( $\vec{T} = \dot{c}(\mathrm{t}) \big/ \left\| \dot{c}(\mathrm{t}) \right\| = \dot{c}(s)$ where the $\left\| . \right\|$ is the length of the vector). The derivative of this vector with respect to arc length is the curvature vector and is normal to the unit tangent for any curve. For a geodesic, it is also normal the surface, and depends only on the surface and its radius ( $\kappa \vec{N}; \kappa = 1/r;$ ). If the radius of the sphere is $r \neq 0$ then $\kappa = 1/r$ . On an ellipsoid, the radius represents the best fitting sphere tangent to the geodesic and ellipsoid at that point.

#### 7.2.2.2 Attribute: interpolation: CurveInterpolation ≡ geodesic

The interpolation for a GeodesicString is always "geodesic".

```
GeodesicString::interpolation:CurveInterpolation="geodesic"
```

The spatial interpolation between any two data points is a geodesic as calculated by the associated GeometricCoordinateSystem, see clause 6.2.5. The basic computational requirements for this interpolation are supplied by the operations in the GeometricCoordinateSystem for "pointAtDistance" in clause 6.2.5.11, "distance" in clause 6.2.5.10, and "bearing" in clause 6.2.5.13.

#### 7.2.2.3 Attribute: controlPoint: DirectPosition [*];

The controlPoint array (equal to the dataPoint array, inherited from Curve) of a Geodesic is a sequence of positions between which the Geodesic is interpolated using geodesics from the geoid or ellipsoid of the coordinate reference system being used. This same array of points also acts as the dataPoint array. The organization of these points is identical to that in Line.

```
GeodesicString::controlPoint:DirectPosition[*]
```

Note For Line and GeodesicString, if any controlPoint lies between the controlPoints on either side on a line or geodesic respectively, then it can be removed without changing the shape of the curve. An exception to this may be dependent on the accuracy of the local system's interpolation mechanisms.

#### 7.2.2.4 Operation: Geodesic (constructor)

The constructor for GeodesicString takes a sequence of points, interpolates using geodesics defined from the geoid (or ellipsoid) of the coordinate reference system being used, and creates the appropriate geodesic string joining them.

```
GeodesicString::GeodesicString(points:DirectPosition[2..*]):Geodes
     icString
```

NOTE If all of the DirectPositions passed to this constructor all lie on the same geodesic curve, then the result is essentially a geodesic segment, and can be represented as a single span. If necessary for operations, intermediate points on the segments can be inserted to increase the density of the dataPoint array, for use in approximation or transformation processing.

© ISO 2013 – All rights reserved

## 7.3 Requirements class: Polynomials

### 7.3.1 Semantics

The equations for lines (7.1.2.2) use functions ( $f_i(t)$ ), usually linear polynomials, in vector equations using the coordinate for controlPoints ( $\vec{P}_i$ ) of the curve. The simplest generalization of this is to replace the linear coefficient functions with polynomials or other functions. This would look like this:

$$\exists f_0, f_1, f_2, ..., f_n \ni f_i : [a,b] \to \mathbb{R};$$

$$c(t) = \sum_{i=0}^{n} f_i(t)\vec{P}_i$$

Another form would use the canonical basis for the coordinate space, i.e.

$$P_i = (\delta_i^j), \quad \text{for } 0 \le \text{i} \le \text{n=dim - 1}$$

$$\text{where} \quad \delta_i^j = \begin{cases} 0 \text{ if } i \ne k \\ 1 \text{ if } i = k \end{cases}$$

In which case the equation reduces to

$$c(t) = (f_0(t), f_1(t), f_2(t), ..., f_{\dim -1}(t))$$

The only requirement for this equation to be meaningful is a Cartesian coordinate system, where the vector space at each point is simply a translation of the tangent space at the origin, and is isomorphic to the original coordinate space. This implies that this sort of system should only be used in engineering coordinate systems, where the underlying coordinates are Cartesian.


© ISO 2013 – All rights reserved

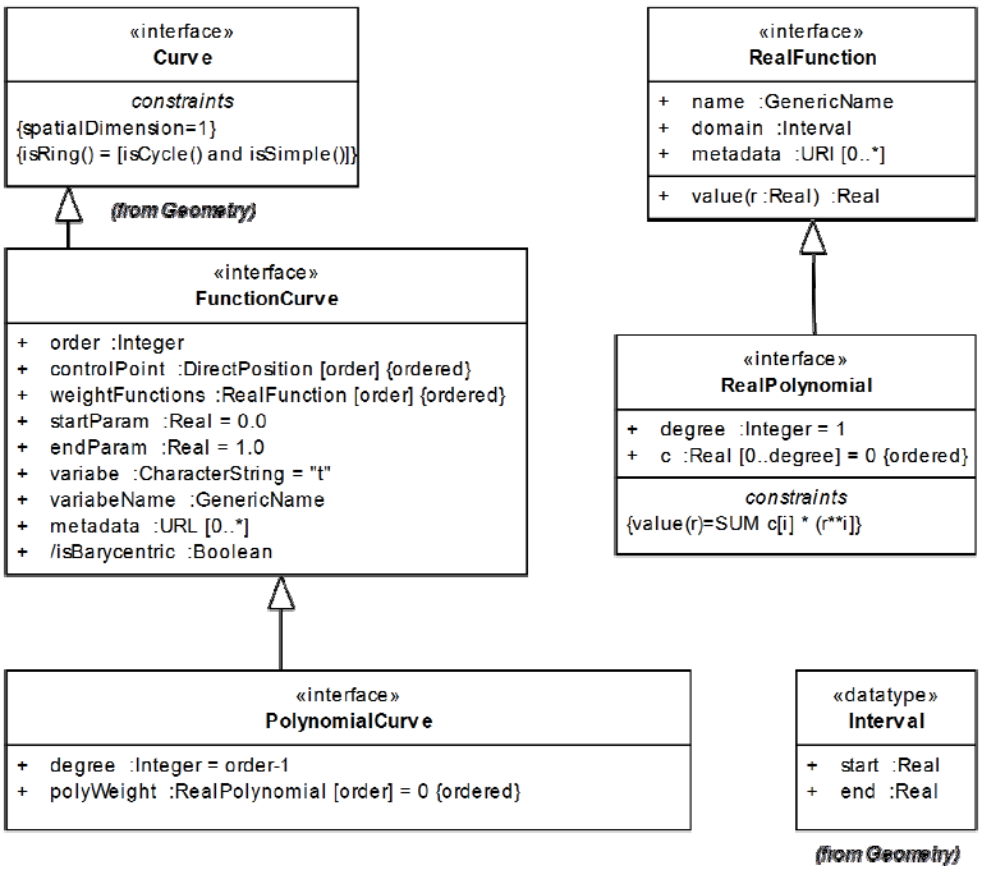



**Figure 20 — Polynomials and Polynomial Curves**

### 7.3.2 Interface: RealFunction

#### 7.3.2.1 Semantics

A RealFunction is any well-defined mapping from an interval of Real numbers to the Real numbers.

#### 7.3.2.2 Attribute: name: GenericName

The attribute "name" of the function is a locally defined identifier for this function.

```
RealFunction:: name: GenericName
```

#### 7.3.2.3 Attribute: domain: Interval

The domain is the interval for which this function is defined.

```
RealFunction::domain:Interal
```

*The function shall be able to return a finite Real for every variable in the domain interval.*

#### 7.3.2.4 Attribute: metadata: URL [0...*]

*The metadata, including definition, should be linked to the function by some number of identifier URL.*

© ISO 2013 – All rights reserved

```
RealFunction::metadata:URL[0…*]
```

#### 7.3.2.5 Operation: value(r: Real): Real

The operation "value" returns the value of the function for a given real in the domain.

```
RealFunction::value(r:Real):Real
```

*The function shall return a valid Real number for each real from the domain of the function. The function may return valid real numbers for a parameter value outside of the domain.*

### 7.3.3 Interface: RealPolynomial

#### 7.3.3.1 Semantics

A RealPolynomial is real function defined by a polynomial. The complete definition of the function is given by the coefficients of terms for each degree up to the maximal degree of the particular polynomial being used. Assuming the variable is $t$ , and the coefficients are stored in an array $c[i],\ i = 0,1,2,...,\text{degree}$, then its value is given by:

$$p(t) = \sum_{i=0}^{\text{degree}} c_i t^i = c_0 + c_1 t + c_2 t^2 ... + c_{\text{degree}} t^{\text{degree}}$$

#### 7.3.3.2 Attribute: degree: Integer

The attribute degree is the degree of the polynomial, i.e. the largest power of the variable used.

```
RealPolynomial::degree: Integer
```

#### 7.3.3.3 Attribute: c: Real

The attribute "c" is the array of the coefficients of the polynomial.

```
RealPolynomial::c [degree+1]: Real
```

### 7.3.4 Interface: FunctionCurve

#### 7.3.4.1 Semantics

A function curve treats the coordinates as vectors in a Euclidean space. This means that either the Figure of the Earth being used is planar or the global coordinate system can be locally treated as $\mathbb{E}^2$ and treated as a vector space or that a local engineering coordinate system is used for an area containing the control points for the curve.

Note One common use is limiting the total area of any interpolation so that the scales of the various directions is more or less constant.

Math The natural engineering space is the tangent space at the initial control point , using the basis for that space made up of the differentials for the coordinate curves (e.g. $[dx,dy]$ for a Euclidean space, or normalize unit vectors in the directions of $[d\varphi,d\lambda]$ for latitude (φ), longitude (λ ), or polar coordinates (distance, bearing)).


© ISO 2013 – All rights reserved

#### 7.3.4.2 Attribute: order: Integer

The attribute "order" is the number of real functions needed, and thus is the number of control points in use.

```
FunctionCurve::order:Integer
```

#### 7.3.4.3 Attribute: controlPoints: DirectPosition [order]

The controlPoints of the curve represent vectors inherited from Curve (6.3.12) are used in groups according to the order of the FunctionCurve.

```
FuctionCurve::controlPoints[order]: DirectPosition[order]
```

One possibility is to use a set of basis vectors for the direct position, i.e.:

$$\delta_i^j = \begin{cases} 1 \text{ if } (i = j) \\ 0 \text{ if } (i \neq j) \end{cases}$$

$$\vec{P}_j = \left\langle \delta_0^j \quad \ldots \quad \delta_i^j \quad \ldots \quad \delta_{\text{dimension-1}}^j \right\rangle$$

$$\vec{c}(u) = \sum_{i=0}^{\text{dimension-1}} f_i(u)\vec{P}_j = \begin{bmatrix} f_0(u) \\ \ldots \\ f_i(u) \\ \ldots \\ f_{\text{dimension-1}}(u) \end{bmatrix}$$

Both forms of function curves will be seen in descriptions of splines curves and surfaces.

#### 7.3.4.4 Attribute: weightFunction: RealFunction [order]

The weight functions control the influence of each control point.

```
FuctionCurve::weightfunctions:RealFunction[order]
```

#### 7.3.4.5 Attribute: startParam: Real = 0.0
#### Attribute: endParam: Real = 1.0

The start and end parameter for the curve combine to define the interval which is the domain of the weight functions.

```
Attribute: startParam: Real = 0.0
Attribute: endParam: Real = 1.0
```

Note The above information is sufficient to define the curve $\vec{c}(t)$, with weight functions $\{f_i(t) \mid 0 \leq i \leq \text{order}\}$ for the values of $t \in [a,b]$, the interval defined by start parameter and end parameter.

© ISO 2013 – All rights reserved

$$\vec{c}(t) = \sum_{i=0}^{order-1} f_i(t)\vec{P}_i; \quad t \in [a,b]$$

#### 7.3.4.6 Attribute: variable: CharacterString ="t" Attribute: variableName: GenericName

The attribute variable, and variable name describe the "symbol" use for the variable and its source. The default variable is "t" and is the constructive parameter defined for Curve.

```
Attribute: variable: CharacterString = "t"
```

#### 7.3.4.7 Attribute: metadata: URL [0...*]

The attribute metadata is a resource link to metadata for the FunctionCurve.

```
Attribute: metadata: URL
```

### 7.3.5 Interface: PolynomialCurve

#### 7.3.5.1 Semantics

A polynomial curve is a FunctionCurve with polynomial weights as in 7.3.1.

Most polynomial curves will be composites where sets of order controlPoints are used for each subsegment (splines, Bezier splines, B-splines and, with a trick in homogeneous coordinates, NURBS). A constructor for a polynomial curve usually takes a number of constraints sufficient to create a solvable system of linear equations for the coefficients.

#### 7.3.5.2 Attribute: degree: Integer

The attribute "degree" is the maximum degree of the real polynomials used.

```
FunctionCurve::degee:Integer=(order-1)
```

Math The degree is usually one less than the number of control points. The larger the degree the more control of the curve shape the designer has. For example, a line segment has two control points, and a degree of one.

In a composite, such as a spline, or NURBS, the order gives the curve designer "local control" in the sense that only the closest "order" control points contribute to a particular value.

#### 7.3.5.3 Attribute: polyWeight: Integer

The attribute array of polynomials "polyWeight" overrides the "weightFunction" array in FunctionCurve by using RealPolynomial (a subtype of RealFunction) instead of RealFunction.

```
PolynomialCurve::weightfunctions =
     PolynomialCurve::RealPolynomial[order]
```

## 7.4 Requirements class: Conics

### 7.4.1 Semantics

The commonality between conic and spiral curves is the use of a control point (center or focus) and a fairly simple representation for a Cartesian 2D coordinate space. The simplest, common mechanism


© ISO 2013 – All rights reserved

for all curves with this property is to "draw" it in a Cartesian plane and then use a projection onto the FoE. For Planes, this required no additional work.

For the curved surfaces (spheres, ellipsoids and geoids) the unifying solution is to choose an Engineering Coordinate system centered on the control point, construct the curve there and then project onto the surface. The common option is to use the tangent space at the control point, express the cure in polar coordinates (bearing and distance). Once the planar curve is defined, mapping it to the figure of the earth used the exponential map, generates a geodesic in the given direction and map to a point along that geodesic at the given distance. This sort of construction is called a geodetic circle.

In all the classes that follow, the basic construction of the curve will use control points and the tangent plane to construct each segment (between consecutive data points). Once the segment is in place, the Engineering CS or tangent plane is projected onto the surface coordinates.

### 7.4.2 Interface: Arc

#### 7.4.2.1 Semantics

An Arc (Figure 21) is similar to a Line except that the interpolation is by circular arcs. The controlPoint array contains the centers for each arc, the dataPoint array the ends of the arcs.

#### 7.4.2.2 Derived attribute: numArc: Integer

The derived attribute "numArc" will be the number of circular arcs in the string.

```
ArcString:numArc:Integer=(controlPoint→length)=(dataPoint→lengt
     h)-1
```

The number of arc in the string will be the length of the controlPoint array and one less than the length of the dataPoint array.

#### 7.4.2.3 Attribute controlPoint [numArc+1]

The inherited controlPoint array will be used as a sequence of centers of the arcs. Beginning at the startPoint of the curve (dataPoint [0]), each controlPoint [i] will be used to center an arc from dataPoint [2i] to dataPoint [2(i+1)]; i = 0, 1, 2… (numArc - 1).

Note The reference system for control point in an implement might be extended to carry the radius and the two azimuths for the circular arc: $\left(\left(position\right), radius, azimth1, azimuth2\right)$. If this is done, the datapoints are defined by the "point at distance" function for the control point controlPoint[i]:

$$controlPo\,\mathrm{int}[i].pointAtDistance\left(radius, azimuth1\right) = dataPoint[2i]\,,$$

$$controlPoint[i].pointAtDistance\left(radius, azimuth2\right) = dataPoint[2i+1] \text{ and}$$

$$controlPoint[i].pointAtDistance\left(radius, azimuth3\right) = dataPoint[2\mathrm{i}+2]\,.$$

The choice of which parameter to store and which to make "derived" is an implementation decision.

#### 7.4.2.4 Attribute dataPoint [2 numArc + 1]

The inherited dataPoint array will be the start, interior and endpoints of the arcs (start and end points being the shared points of two consecutive arcs in the string.

© ISO 2013 – All rights reserved

#### 7.4.2.5 Attribute radius [numArc] Vector

Each radius [i] vector is in the tangent space of the corresponding controlPoint [i]. If the space is 3D, the radius vector, and the two endpoints of the arc will determine the plane of the arc. In all cases, the 3 points, two from the dataPoint array and the end of the vector from the control point determine the sense of the arc, starting at the first dataPoint [i], passing through the point determined by the vector (using the operation "pointAtDistance(controlPoint [i], radius [i])") and terminating at the second dataPoint [i+1]. Each arc should be less than 360° to allow for non-ambiguous sense of the arc's rotation.

#### 7.4.2.6 Datatype: ArcConstructor

The arc uses common supertype datatype with circle as input into the default constructor. The datatype ArcConstructor has two concrete subtypes, ArcByCenter and ArcByBulge. ArcByCenter contains:

- a list of centers as control points: `center[]:DirectPosition`
- a list of rotational directions for the arc: `rotation[]:Rotation`
- a list of radius as a Vector to the first dataPoint: `radius[]: Vector`
- a list of datapoints: `dataPomt[]:DirectPosition`

Each arc will span two consecutive dataPoints, with the first shared with the previous arc and the last shared with the next. So, the dataPoint array will be one longer than the center array.

The rotation of the arc will be with respect to the two radius vectors (which should not be equal nor parallel. Rotational direction is observed from the top of the cross product vector of two consecutive radius vectors associated to the same center point.

```
center→length+1 = dataPoint→length
radius[i].CrossPoduct(radius[i+1])≠0
```

ArcByBulge contains:

- the number of arcs subsegments
- a list of datapoints: dataPoint[numArc+1]:DirectPosition; the ends of the arc
- a list of bulges, directed distance, "-" for leftward bulge producing a clockwise rotation, "+" for rightward bulge producing a counterclockwise rotation.

#### 7.4.2.7 Constructor ArcString (a: ArcConstructor): ArcString

The only required constructor for ArcString simply supplies values for controlPoint, dataPoint and radius arrays.


© ISO 2013 – All rights reserved

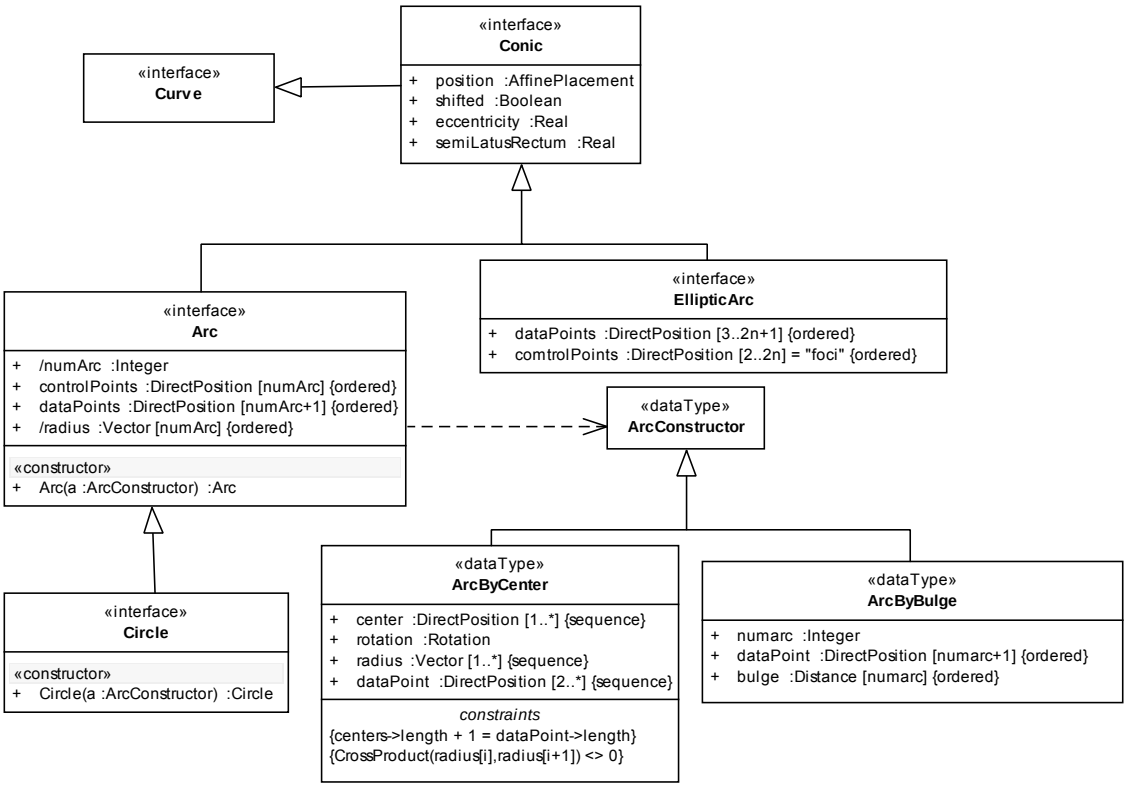



**Figure 21 — Conics, Arcs and Circles**

### 7.4.3 Interface: Circle

#### 7.4.3.1 Semantics

The interface Circle is the same as that for ArcString, but is assumed each arc has the same center, the same radius length and to be closed to form a full circle. The "start" and "end" bearing are equal and shall be the bearing for the first and last dataPoint listed.

At a minimum because the arc must be less that 360°, the controlPoint array will at least two-long with the same point in each position (the center of the circle). The dataPoint array and the points determined by the radius vectors will give an non-ambiguous sense of the orientation of the circular curve. The "default" configuration would have $dataPoint[0]$. $controlPoint[0]$, and $dataPoint[1]$ as 3 points on a common geodesic diameter of the circle, and the vectors $radius[0] = radius[2] = -radius[1]$ perpendicular to the geodesic at the center fixing the rotation of the circle (starting at $dataPoint[0]$ rotating towards $radius[0]$ and then completing the circle through $dataPoint[1]$, and back to $dataPoint[0]$ .

#### 7.4.3.2 Constructor Circle((a: ArcConstructor): Circle

The circle inherits the constructor from ArcString. I more than

```
Circle((a: ArcConstructor):Circle
```

© ISO 2013 – All rights reserved

#### 7.4.3.3 Constructor: Circle (center: DirectPosition, radius: Distance, startAngle: Bearing): Circle

The circle also has a simpler constructor that only needs the center, radius and start angle (direction is from the rotation to the radius must be less than 180°.

```
Circle(center:DirectPosition,radius:Distance,startAngle:Bearing):
     Circle
```

### 7.4.4 Interface: Conic

#### 7.4.4.1 Semantics

The type Conic (Figure) represents any general conic curve. Any of the conic section curves can be canonically represented in polar co-ordinates ($\rho$, $\theta$) as:

$$\rho = \frac{ed}{\left(1 + e\cos\theta\right)} \quad for -\frac{\pi}{2} \le \theta \le +\frac{\pi}{2}$$

and e is the eccentricity and d is the distance to the directrix.

where

$P$ is semi-latus rectum;

$e$ is the eccentricity.

This gives a conic with focus at the pole (origin), and the vertex on the conic nearest this focus in the direction of the polar axis, $\varphi = 0$ (at $(\rho,\varphi) = \left(\left(\frac{P}{1+e}\right),0\right)$ in polar coordinates). For $e = 0$, this is a circle. For $0 < e < 1$, this is an ellipse. For $e = 1$, this is a parabola. For $e > 1$, this is one branch of a hyperbola.

These generic conics can be viewed in a 2-dimensional Cartesian parameter space ($u$, $v$) given by the usual coordinate conversions $u = \rho\cos(\varphi)$ and $v = \rho\sin(\varphi)$. We can then convert this to a 3D coordinate reference system by using an affine transformation, $(u, v) \rightarrow (x, y, z)$ which is defined by:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} u_x & v_x \\ u_y & v_y \\ u_z & v_z \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

This gives us $\varphi$ as the constructive parameter. The DirectPosition given by ($x_0$, $y_0$, $z_0$) is the image of the origin in the local coordinate space ($u$, $v$).

Alternatively, the origin may be shifted to the vertex of the conic as

$u' = \rho\cos(\varphi) - \left(P/_{1+e}\right)$ and $v' = \rho\sin(\varphi)$

and v can be used as the constructive parameter (see definition at Curve, 6.3.12 ).


© ISO 2013 – All rights reserved

In general, conics with small eccentricity and small P use the first or "central" representation. Those with large eccentricity or large P tend to use the second or "linear" representation.

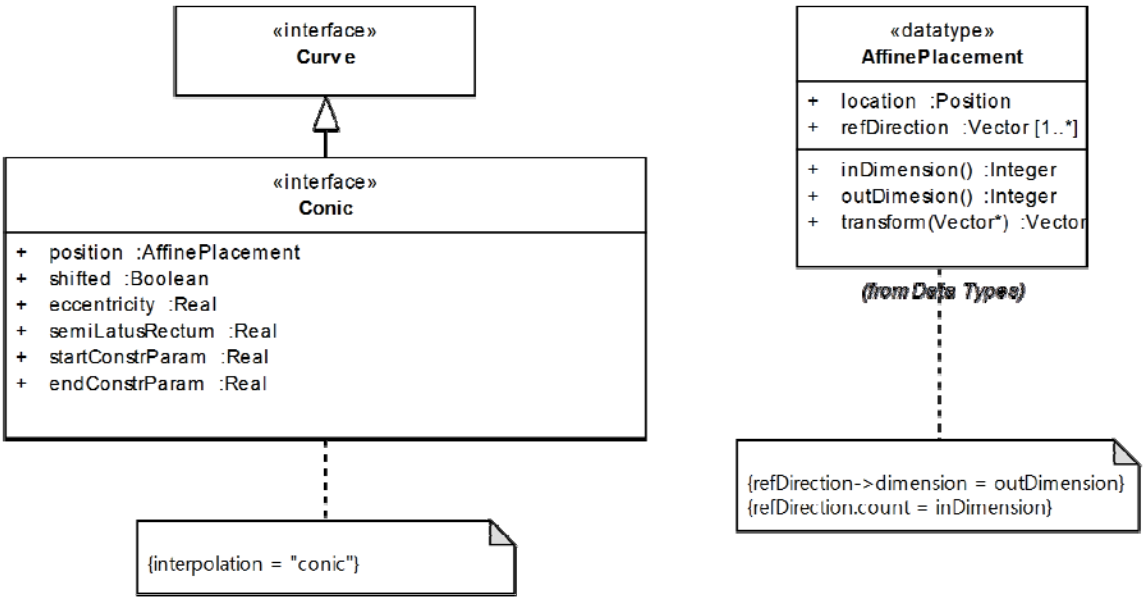



**Figure 22 — Conics and placements**

#### 7.4.4.2 Attribute: position: AffinePlacement

The attribute "position" will be an affine transformation object that maps the conic from parameter space into the coordinate system of the conic as a GeometryObject. This affine transformation is given by the formulae in the previous clause.

```
Conic::position:AffinePlacement
```

#### 7.4.4.3 Attribute: shifted: Boolean

The attribute "shifted" is FALSE if the affine transformation is used on the ($u$, $v$) and TRUE if the affine transformation is applied to the shifted parameters ($u$', $v$'). This controls whether the focus (shifted = FALSE) or the vertex (shifted = TRUE) of the conic is at the origin in parameter space.

```
Conic::shifted:Boolean
```

#### 7.4.4.4 Attribute: eccentricity: Real

The attribute "eccentricity" is the value of the eccentricity parameter "$e$" used in the defining equation above. It controls the general shape of the curve, determining whether the curve is a circle, ellipse, parabola, or hyperbola.

```
Conic::eccentricity:Real
```

#### 7.4.4.5 semiLatusRectum

The attribute "semiLatusRectum" is the value of the parameter "$P$" used in the defining equation above. It controls how broad the conic is at each of its foci.

© ISO 2013 – All rights reserved

```
Conic::semiLatusRectum:Real
```

#### 7.4.4.6 startConstrParam, endConstrParam

The "startConstrParam" and "endConstrParam" indicate the parameters used in the constructive parameterization, given in 6.4.19.1, for the startPoint and endPoint respectively:

```
Conic::startConstrParam:Real
Conic::endConstrParam:Real
 Conic:
constrParam(startConstrParam)=startPoint();
constrParam(endConstrParam)=endPoint();
```

There is no assumption that the startConstrParam is less than the endConstrParam, but the parameterization must be strictly monotonic (strictly increasing, or strictly decreasing).

### 7.5 Requirements class: Spirals

### 7.5.1 Mathematical background: cures and curvature

A spiral curve is a $C^3$ curve with strictly monotonic curvature. Why this is important is traced to the formula in physics for centrifugal force

$$a = v^2 / r$$

Where "*a*" is acceleration, "*v*" is "speed" (giving velocity along the curve) and "*r*" is the radius of the path. For a curve, the "r" is replaced by an expression of curvature:

$$\kappa = 1/r \;\Rightarrow\; a = \kappa v^2$$

This means that the lateral force on a moving object (assuming a velocity) is proportional to the curvature of the path. The tighter the curve, the larger the curvature, the stronger the side force. By controlling the curvature of the road or railroad, the designer can control the side force on the vehicle (car or train) and thus control behavior, limiting the chance of a "spinout" or "derailment."

The important fact that needs to be known is the two versions of the fundament theorem for curves. These theorems equate the definition of a curve with the definition of its curvature ( $\kappa(t)$ ) which describes how the direction of the curve changes in time and torsion ( $\tau(t)$ ) which describes how the curve twists (changes plane) in 3D through time. Note that these are valid only where the calculus can be used in the Cartesian coordinate space of the proper dimension (the physics is usually done in an Engineering coordinate system where the classic Newtonian forces fit the classic Newtonian equations).

For example, in a clothoid spiral, curvature is changes linear with arc length, and so with a constant acceleration (deceleration) on an opening (closing) curve, the side force is constant.

**Fundamental Theorem of Plane Curves (in $\mathbb{E}^2$):**


© ISO 2013 – All rights reserved

Let $I$ be an interval $I=(a,b)\subseteq\mathbb{R}$ and let $\hat{\kappa}:I\subset\mathbb{R}\to\mathbb{R}$ be a continuous function. Then there exist a curve $c:I\to\mathbb{R}^2$ such that $\|\dot{c}(s)\|=1$ and the curvature of $c$ is $\hat{\kappa}$. Any two such curves will differ by a translation and a 2D rotation.

**Fundamental Theorem of Space Curves (in $\mathbb{E}^3$):**

Let $I$ be an interval $I=(a,b)\subseteq\mathbb{R}$ and let $\hat{\kappa}:I\to\mathbb{R}$ with $\hat{\kappa}>0$ and $\hat{\tau}:I\to\mathbb{R}$ be continuous functions. Then there is a smooth curve $c:I\to\mathbb{R}^3$ such that $\|\dot{c}(s)\|=1$ and the curvature and torsion of $c$ are $\hat{\kappa}$ and $\hat{\tau}$ .Any two such curves will differ by a translation and a 3D rotation.

To understand what these mean and how they will be used in this standard, some fundamental concepts of differential geometry need to be explained.

This international standard uses two parameterizations for each curve interpolation type. The first is a constructive parameter $t$ that is chosen by the implementation for their algebraic convenience. The second is arc length $s$ which is useful in defining some differential geometric concepts. Statements in the text that do not differentiate between the two will be true for either choice.

For example, if $c$ is a curve, then $c(t)$ is the curve parameterized by $t$, a constructive parameter, and $c(s)$ is the curve parameterized by s, arc length from some fixed start point, positive after that point, negative before it. To maintain the orientation of $c$, both variables must increase in the same direction along the curve.

Because many curves are defined using standard calculus in a standard, flat, Euclidean space $\mathbb{E}^2$, a local engineering CRS is required, which can then transform to the CRS of the data set. The most universally applicable Engineering CRS is the tangent plane at a particular point, which maps back to the Figure of the Earth using the exponential map.

Let $c(t)=(x(t),y(t))$ be a curve, which is represented by a function $c:\mathbb{E}^1(\mathbb{R}^1)\to\mathbb{E}^2$. The variable $t$ is any continuous parameterization of the curve. The derivative

$$\dot{c}(t)=(\dot{x}(t),\dot{y}(t))$$

is a vector in the tangent space of vectors at $c(t)$. The length of the derivative tangent is given by the Pythagorean Theorem ($t$ and $s$ increase in the same direction of the curve)

$$\|\dot{c}(t)\|=\left(\sqrt[2]{\dot{x}^2(t)+\dot{y}^2(t)}\right)=\frac{ds}{dt}.$$

If $c(t)$ is written in polar coordinates where r and θ are functions of t

$$\exists(r(t),\theta(t))\ni c(t)=r(t)(\cos\theta(t),\sin\theta(t)),$$

where $r(t)=\|c(t)\|$, $x(t)=r(t)\cos\theta(t)$ and $y(t)=r(t)\sin\theta(t)$.

© ISO 2013 – All rights reserved

The unit tangent $\vec{\mathrm{T}}$ is

$$\vec{\mathrm{T}}(t) = \dot{c}(t)\frac{dt}{ds} = \frac{\dot{c}(t)}{\left\|\dot{c}(t)\right\|} = (\cos\theta, \sin\theta)$$

The vector $\vec{\mathrm{N}}$ perpendicular to the curve, and point leftward so that $\left(\vec{\mathrm{T}}, \vec{\mathrm{N}}\right)$ is right handed frame, is:

$$\vec{\mathrm{N}}(t) = (-\sin\theta, \cos\theta)$$

Let $c(s) = \left(x(s), y(s)\right)$ be the same curve parameterized by arc length, which is also a function $c : \mathbb{E}^1\left(\mathbb{R}^1\right) \to \mathbb{E}^2$. The variable $s$ is the arc length from some fixed "start point" on the curve. The derivative

$$\dot{c}(s) = \left(\dot{x}(s), \dot{y}(s)\right)$$

is a vector in the tangent space of vectors at $c(s)$.

Because $s$ is arc length, $\left\|\dot{c}(s)\right\| \equiv 1$ and $\dot{c}(s)$ can be expressed in terms of the same $\theta$, where $\theta(t) = \theta(s)$:

$$\exists \theta(s) \ni \dot{c}(s) = \vec{\mathrm{T}}(s) = (\cos\theta, \sin\theta)$$

Since the two parameterizations traverse the same geometry, there is a function

$$t \to s \ni c(t) = c(s)$$

If $\frac{dt}{ds} \neq 0$, then $\dot{c}(s) = \dot{c}(t) / \left\|\dot{c}(t)\right\| = \vec{\mathrm{T}}(t) = \vec{\mathrm{T}}(s)$; and similarly

$$\vec{\mathrm{N}}(s) = \vec{\mathrm{N}}(t) = (-\sin\theta, \cos\theta).$$

Since s is arc length, the speed (magnitude of velocity in the direction of the tangent) is constant and the second derivative is perpendicular to the curve, i.e. perpendicular to the tangent $\vec{\mathrm{T}}$.

$$\ddot{c}(s) = \frac{d}{ds}\dot{c}(s) = \frac{d}{ds}\vec{\mathrm{T}}(s) = \frac{d}{ds}(\cos\theta, \sin\theta) = \frac{d\theta}{ds}(-\sin\theta, \cos\theta) = \kappa(s)\vec{\mathrm{N}}(s)$$

The scalar function $\kappa(s)$ is the curvature of $c$ at $c(s)$. The above equation also gives a very important connection between $\theta$ and $\kappa$:


© ISO 2013 – All rights reserved

$$\frac{d\theta(s)}{ds} = \kappa(s)$$

Similarly when the normal, has a similar relation:

$$\frac{d}{ds}\vec{\mathrm{N}}(s) = \frac{d}{ds}(-\sin\theta, \cos\theta) = -(\cos\theta, \sin\theta)\frac{d\theta}{ds} = -\kappa(s)\vec{\mathrm{T}}(s)$$

So, for planar curves (2D Frenet–Serret formulas):

$$\dot{T}(s) = \kappa N$$
$$\dot{N}(s) = -\kappa T$$

In terms of $t$, the curvature $\kappa$ can be expressed:

$$\kappa(t) = \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\left(\sqrt[2]{\dot{x}^2 + \dot{y}^2}\right)^3} = \frac{\pm\dot{c}(t) \times \ddot{\mathrm{c}}(t)}{\dot{c}(t)^3}$$

Assuming that $\kappa(s) \neq 0$ then, the best fitting circle for the curve at $c(s)$, has a radius equal to the inverse of the curvature, $R(s) = 1/\kappa(s)$ and its center offset from the curve in the direction $\kappa(s)\vec{\mathrm{N}}(s)$. If the curve bends to its left, $\kappa(s) > 0$, and if it bends to the right $\kappa(s) < 0$. Constant curvature implies that the curve is either a line or a circular arc:

$$[\kappa \equiv 0] \Rightarrow [c \in Line]$$

$$\left[\kappa \equiv \frac{1}{r} \neq 0\right] \Leftrightarrow [c \in Circular\ Arc\ with\ radius\ |r|]$$

The proof in 3-space ($\mathbb{E}^3$)is similar except that it requires solving a set of differential equations, and uses a function to describe the twisting of the curve in three space, and used the 3D version of the Frenet–Serret formulas:

$$\dot{T}(s) = \kappa N$$
$$\dot{N}(s) = -\kappa T + \tau B$$
$$\dot{B}(s) = -\tau N$$

### 7.5.2 Creating a spiral from a curvature function

The relationship between curvature and arc length derived above is:

$$\frac{d\theta}{ds} = \kappa(s)$$

© ISO 2013 – All rights reserved

$$\theta(s) = \theta(0) + \int_0^s \kappa(\xi)\, d\xi$$

$$\dot{c}(s) = \tau(s) = (\cos\theta, \sin\theta)$$

$$c(s) = c(0) + \int_o^s (\cos\theta(\xi), \sin\theta(\xi))\, d\xi$$

An Euler spiral, clothoid or Cornu spiral is a curve whose curvature changes linearly with its curve length. So there is a constant such that:

$$\exists a \in \mathbb{R} \ni \kappa(s) = as$$

Thus

$$\frac{d\theta}{ds} = \kappa(s) = as$$

So

$$\theta(s) = \frac{a}{2}s^2 + \theta(0) = \int_0^s \kappa(\xi)\, d\xi$$

Assuming we start with the origin with a tangent the positive x-axis ($\theta = 0$):

$$\dot{c}(s) = (\cos\frac{a}{2}s^2, \sin\frac{a}{2}s^2) = \left( \cos\left( \int_0^s \kappa(\xi)\, d\xi \right), \sin\left( \int_0^s \kappa(\xi)\, d\xi \right) \right)$$

For the specific case of the clothoid, we can define:

$$C(s) = \int_0^s \cos\left( \frac{a\xi^2}{2} \right) \mathrm{d}\xi \text{ and } S(s) = \int_0^s \sin\left( \frac{a\xi^2}{2} \right) \mathrm{d}\xi$$

Then, using the Taylor series expansions for sine and cosine:

$$\sin x = \sum_{k=0}^{\infty} \frac{(-1)^k x^{1+2k}}{(1+2k)!} \text{ and } \cos x = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{(2k)!}$$

gives with substitution and polynomial integration:


© ISO 2013 – All rights reserved

$$c(s) = \int_o^s \left( \sum_{k=0}^{\infty} \frac{(-1)^k \left(\frac{a\xi}{2}\right)^{2k}}{(2k)!}, \sum_{k=0}^{\infty} \frac{(-1)^k \left(\frac{a\xi}{2}\right)^{1+2k}}{(1+2k)!} \right) d\xi$$

$$= \left( \sum_{k=0}^{\infty} \frac{(-1)^k \left( \left(\frac{a}{2}\right)^{2k} s^{2k+1} \right)}{(2k+1)(2k)!}, \sum_{k=0}^{\infty} \frac{(-1)^k \left( \left(\frac{a}{2}\right)^{1+2k} s^{2(k+1)} \right)}{2(k+1)(1+2k)!} \right).$$

This series converges fairly quickly for small s, which is quite sufficient for the types of transition curves for which spirals are most commonly used.

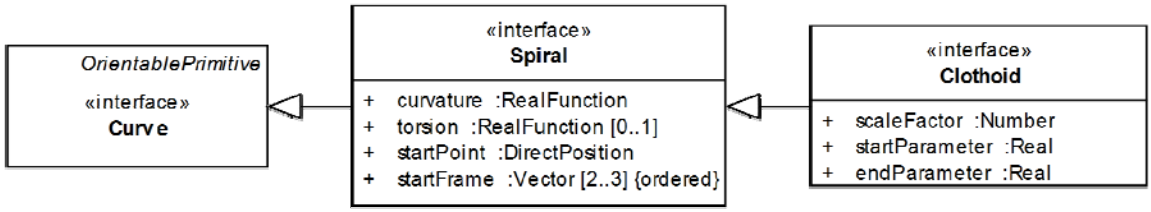



**Figure 23 — Spirals**

### 7.5.2.1 Interface: Spiral

#### 7.5.2.1.1 Semantics

The general spiral is specified by its curvature function in $\mathbb{E}^2$ and by its curvatuve and torsion functions in $\mathbb{E}^3$.

#### 7.5.2.1.2 Attribute: curvature: RealFunction

The attribute curvature contains (or references) the curvature function of the spiral.

*The curvature function shall be the curvature function of the curve in terms of its arc length.*

*The domain of the curve shall be the domain of the curvature function.*

#### 7.5.2.1.3 Attribute: torsion: RealFunction

The optional attribute torsion contains (or references) the torsion function of the spiral.

*If the torsion function is not specified, the curve is planar and only required curvature. If the torsion function is specified (and non-zero somewhere), the the spiral will not be planar.*

*The domain of the torsion function shall be the same as domain of the curvature function.*

© ISO 2013 – All rights reserved

#### 7.5.2.1.4 Attribute: startPoint: RealFunction

The startPoint of the curve will be the intial position of the curve. It is inherted from curve, but is replicated here because of its importance in calculating a spiral.

#### 7.5.2.1.5 Attribute: startFrame: Vector[2...3]

The startFrame of the curve is a orthonormal frame of 2 or 3 vectors. Each one is a unit vector, each is orthogonal to the others and together they form a right handed frame.

*It the start frame is 2D, then the curve is a planar spiral, with the first vector is its tangent, and the second is its normal.*

*If the start frame is 3D, then the curve is not planar, the torsion function is not everywhere zero.*

### 7.5.2.2 Interface: Clothoid

#### 7.5.2.2.1 Semantics

Clothoids are used almost exclusively in road construction. Their shape works well with the physics of a vehicle following a curve. These are specialty curves, and are in a separate package because of their limited use implies a separable conformance test for them.

Clothoid (Figure 23) implements the clothoid (or Cornu's spiral), which is a planar curve whose curvature is a fixed function of its length. In suitably chosen co-ordinates it is given by Fresnel's integrals:

$$x(t) = \int_0^t \cos\left(\frac{A\tau^2}{2}\right) d\tau \text{ and } y(t) = \int_0^t \sin\left(\frac{A\tau^2}{2}\right) d\tau$$

.

See Kostov **Error! Reference source not found.** in the bibliography for further properties of and methods associated with clothoid curves and piecewise clothoid curves.

This geometry is mainly used as a transition curve between curves of type straight line/circular arc or circular arc/circular arc. With this curve type it is possible to achieve a $C^2$-continous transition between the above mentioned curve types. One formula for the clothoid is $A^2 = R*t$ where A is a constant, *R* is the varying radius of curvature along the curve and *t* is the length along the curve and given in the Fresnel integrals.

#### 7.5.2.2.2 Attribute: position: DirectPosition

The attribute "position" is an affine mapping that places the curve defined by the Fresnel Integrals into the coordinate reference system of this object.

```
Clothoid::position:AffinePlacement
```

#### 7.5.2.2.3 Attribute: scaleFactor: Number

The attribute "scaleFactor" gives the value for A in the equations above.


© ISO 2013 – All rights reserved

```
Clothoid::scaleFactor:Number
```

#### 7.5.2.2.4 Attribute: startConstrParam: Real

The attribute "startConstrParam" inherited from Curve is used here as the arc length distance from the inflection point that will be the start point for this curve segment. This shall be lower limit "*t*" used in the Fresnel integral and is the value of the constructive parameter of this curve segment at its start point. The "startConstrParam" can be either positive or negative. The parameter "*t*" acts as a constructive parameter.

```
Curve::startParameter:Real
```

NOTE If 0.0 (zero), lies between the startConstrParam and endConstrParam of the clothoid, then the curve goes through the clothoid's inflection point, and the direction of its radius of curvature, given by the second derivative vector, changes sides with respect to the tangent vector. The term "length" for the parameter "*t*" is applicable only in the parameter space and its relation to arc length after use of the placement, and with respect to the coordinate reference system of the curve is not deterministic.

#### 7.5.2.2.5 Attribute: endConstrParam: Real

The attribute "endConstrParam" inherited from Curve is used here as the arc length distance from the inflection point that will be the end point for this curve segment. This shall be upper limit "*t*" used in the Fresnel integral and is the constructive parameter of this curve segment at its end point. The "endConstrParam" can be either positive or negative.

```
Clothoid::endParameter:Real
```

Ed: The following two sections reference spiral curves in *ISO/IEC 13249-3:2006(E) - Information technology — SQL Multimedia and Application Packages - Part 3: Spatial*. These will have formal definitions which are agreed to by the group doing SQL/MM

#### 7.5.2.3 Interface: Bloss spirals

#### 7.5.2.3.1 Semantics

#### 7.5.2.4 Interface: Biquadratic spirals

#### 7.5.2.4.1 Semantics

## 7.6 Requirements class: Spline Curve

### 7.6.1 Semantics

Spline curves (see Figure 24) are mathematical approximations to the classic draftsman spline. This was a flexible ruler that could be bent into position through the use pins or weights attached to it. Physically, such a device created a curve that was piecewise a cubic polynomial. Later development of splines concentrated on their mathematical and computational properties, such as how they reacted under various types of transformations, issues about local control was and how difficult they were to calculate. All splines share the property that they can be represented by parametric functions that map into the coordinate system of the geometric object that they will represent. Spline Curves come in essentially two forms: interpolants and approximants. Both forms are examples of the FunctionCurve described in Clause 7.3.

Fitted or interpolating splines ("interpolant") are exact calculations, usually for each coordinate offset separately, that create curves in each coordinate space that passes through each of the given

© ISO 2013 – All rights reserved

control points. In general, the curves are defined by their datapoints with extra conditions at boundary points (the data points at either end of the segment) and the level of continuity. For example, a cubic spline changes formula at each data point, passes through each data point, is continuous and has a smooth tangent at each point. This is the best that can be done, since a cubic polynomial only has 4 coefficients to match the 4 criteria (value at each end, and continuity of tangent at each end). The design of a cubic spline will also allow for the choice of the tangent directions at the start and end point.

The second types ("approximants") only approximate the control points. These splines use sets of real valued functions that form a "partition of unity." Such functions are all defined on a single common domain, are always non-negative in their values, and always sum as a complete set to 1.0 for their entire domain. These functions are then used in vector equations for those coordinate offset to which they will be applied, each associated to a control point, so that the tracing of the curve is a weighted average based on the partition of unity. Since the spline curve function are all non-negative weighted sum of control points, it value always lies in the convex hull of the control points whose weight function is currently non-zero (this is called to local convexity property), and thus always in the convex hull of the control points (the global convexity property). Since such functions are defined in vector form, they can generally be used in any target dimension coordinate system.

Most partitions of unity are originally defined as piecewise polynomials, which when applied to homogeneous coordinates and projected down to standard coordinates give us a related set of rational functions which are still a partition of unity, and hence another type of spline. Approximants have nice properties involving easy of representation, ease of calculation, smoothness, and some form of convexity. The do not always pass through the control point, but if the control point array is dense enough, the local properties will force a good approximation of them, and will give a well-behaved curve in terms of shape and smoothness. In some cases, generally splines of degree 1 (line strings), and in the case of bilinear and bicubic surfaces, approximants do pass through their control points.

Note: For polynomial curves, the system of equations to be solved will always be a linear system with variable the coefficients of the various defining polynomials, and the matrix solution depends on the constraints chosen. Given a new set of data points, the solution can be reused by replacing the old data point values with the new. This will be important in the description of TensorSpline surfaces

### 7.6.2 Interface: Knot

#### 7.6.2.1 Semantics

The knots are values from the domain of the constructive parameter space for splines, curves, surfaces and solids. Each knot sequence is used for a dimension of the spline's parameter space $\overrightarrow{k_i} \in \{u_0, u_1, u_2 ...\}$. Thus, in a surface spline, there will be two knot sequences, one for each parameter in $\overrightarrow{k_{i,j}} = \left(\mathrm{u}_i, v_j\right)$.

In the knot sequence, a knot can be repeated (affecting the underlying spline formulae). The number of repetitions is the multiplicity of the knot. The internal representation of the knot sequence has two equivalent representations. The first is a simple sequence, with repetitions of each knot for its multiplicity:

$$T = \{t_0, t_1, ..., t_{m-1}\} \text{ with } t_i \le t_{i+1}$$

The second form is each knot (a real, in $\mathbb{R}$) is distinct and accompanied by a multiplicity(an integer, in $\mathbb{Z}$):


© ISO 2013 – All rights reserved

$$k_i = (t \in \mathbb{R}, m \in \mathbb{Z})$$

Either storage format is acceptable, but the first is more commonly used in the defining formulae for the various splines.

#### 7.6.2.2 Attribute: value: Real

The attribute "value" is the value of the parameter at the knot of the spline. The sequence of knots shall be a non-decreasing sequence. That is, each knot's value in the sequence shall be equal to or greater than the previous knot's value. The use of equal consecutive knots is normally handled using the multiplicity.

```
Knot□value:Real
```

#### 7.6.2.3 Attribute: multiplicity: Integer

The attribute "multiplicity" is the multiplicity of this knot used in the definition of the spline. The semantics of the multiplicity is explained in the description of the various splines.

```
Knot□multiplicity:Integer
```

### 7.6.3 CodeList: KnotType

A B-spline is uniform if and only if all knots are of multiplicity one and they differ by a positive constant from the preceding knot. A B-spline is quasi-uniform if and only if the knots are of multiplicity (degree+1) at the ends, of multiplicity one elsewhere and they differ by a positive constant from the preceding knot. This code list is used to describe the distribution of knots in the parameter space of various splines. Some possible values are:

1. Uniform (uniform): knots are equally space, all multiplicity 1.
2. Non-uniform (nonUniform): knots have varying spacing and multiplicity.
3. Quasi-Uniform (quasiUniform): the interior knots are uniform, but the first and last have multiplicity one larger than the degree of the spline (p+1).
4. Piecewise Bézier (piecewiseBezier): the underlying spline is formally a Bezier spline, but knot multiplicity is always the degree of the spline except at the ends where the knot degree is (p+1). Such a spline is a pure Bézier spline between its distinct knots.

Some potential values of the code list KnotType are

```
KnotType::
   uniform
   nonUniform
   quasiUniform
   piecewiseBezier
```

This knot type is used for informational purposed, and it should be set in a manner consistent with the actual knot sequences.

© ISO 2013 – All rights reserved

### 7.6.4 CodeList: SplineCurveForm

The code list “SplineCurveForm” is used to indicate which sort of curve is approximated by a particular B-spline.

Some potential values of the code list SplineCurveForm are:

1. Polyline, (polyline): a connected sequence of line segments represented by a one degree spline (a line string).
2. Circular Arc (circularArc): an arc of a circle or a complete circle.
3. Elliptical Arc (ellipticalArc): an arc of an ellipse or a complete ellipse.
4. Parabolic (parabolicArc): an arc of a finite subsegment of a parabola.
5. Hyperbolic (hyperbolicArc): an arc of a finite length of one connected branch of a hyperbola.

```
SplineCurveForm::
   polyline
   circularArc
   ellipticalArc
   parabolicArc
   hyperbolicArc
```

This will be used for informational purposes, and should be consistent with the other properties of the spline

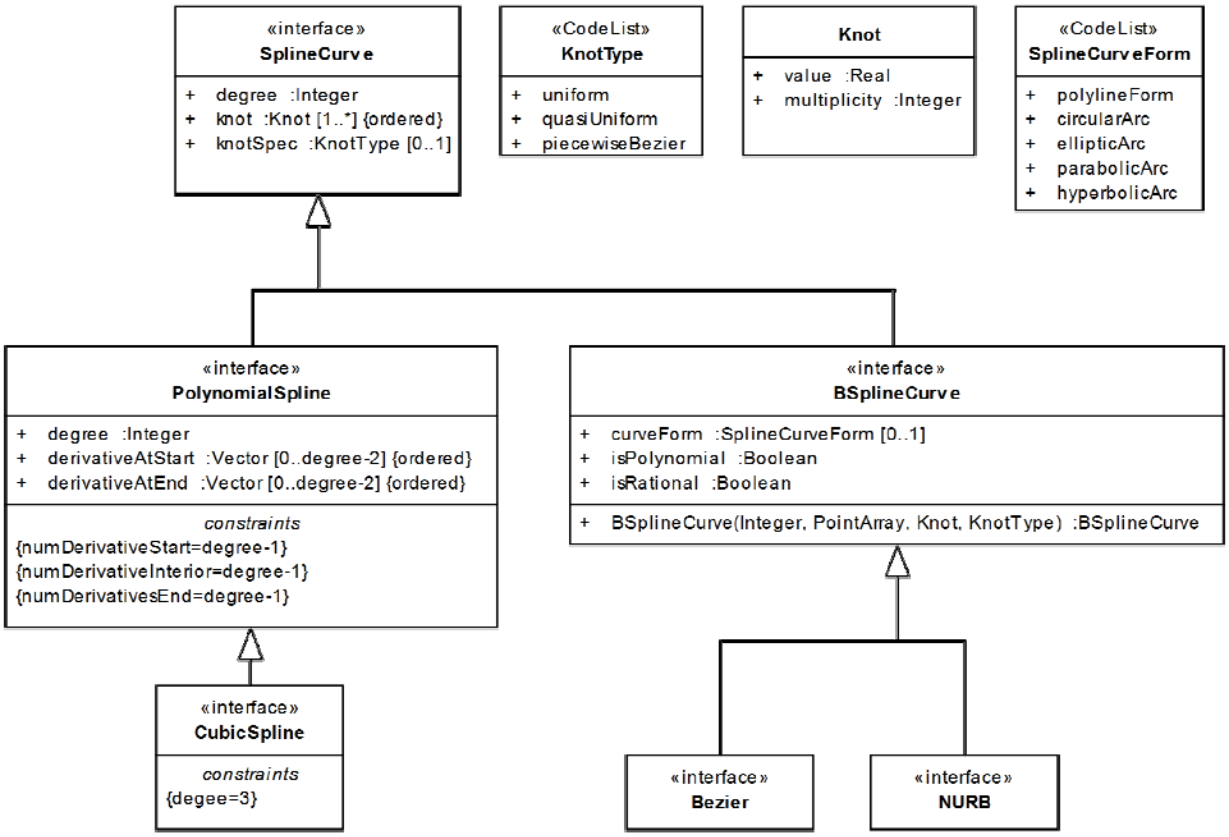



**Figure 24 — Spline Curves**


© ISO 2013 – All rights reserved

### 7.6.5 Interface: SplineCurve

#### 7.6.5.1 Semantics

SplineCurve (Figure 24) acts as a root for subtypes of Curve using some version of spline, either using polynomial or rational functions; see clause 7.3.

#### 7.6.5.2 Attribute: knot: Knot [*]

The attribute "knot" shall be the array of distinct knots, each of which will define a value in the parameter space of the spline, and will be used to define the spline basis functions.

The knot data type holds information on knot multiplicity (Clause 7.6.2). Repetitions in the knot values will be distinguished through use of this multiplicity, and so the parameter values in this array will be strictly increasing, i.e.

$$\forall i, 0 \le i < knot.length : knot[i].value < knot[i+1].value$$

```
SplineCurve::knot:Knot[*]
```

For each knot will be associated to the corresponding data point as follows:

$$c(\mathrm{k}_i) = \mathrm{dataPoint[i]}$$

For an interpolating spline, the data points are the control points. For an approximating spline the data points are calculated from the control points, and the each data point will be generally near its corresponding control point.

#### 7.6.5.3 Attribute: degree: Integer

The attribute "degree" shall be the degree of the polynomials used for defining the interpolation in this SplineCurve. Rational splines will have this degree is the limiting degree for both the numerator and denominator of the rational functions being used for the interpolation.

```
SplineCurve::degree:Integer
```

NOTE In some sense, the multiplicity of a knot counteracts the degree of the spline in determining smoothness of a curve at the knot values, so knot multiplicity will always be less than or equal to degree. It this is not done, the curve will likely to be discontinuous unless a control point is repeated enough times.

#### 7.6.5.4 Attribute: isRational: Boolean

The attribute "isRational" indicates that the spline uses rational functions to define the curve. This is done by creating a polynomial spline on homogeneous coordinates, and projecting back to regular coordinates when all calculations are done. If the weights of all of the control points are equal, then the spline is equivalent to the corresponding polynomial spline after the projection. This mechanism is described in Annex B.5.

```
SplineCurve::isRational: Boolean
```

In a rational spline, each control point is given a weight.

© ISO 2013 – All rights reserved

$$\begin{aligned}\hat{P}_i &= (w_i x_0, w_i x_1, w_i x_2, ..., w_i x_{n-1}, w_i) \\ &= (x_0, x_1, x_2, ..., x_{n-1}, 1) \\ &\rightarrow (x_0, x_1, x_2, ..., x_{n-1}) = P_i\end{aligned}$$

If the $\hat{P}_i$ are the poles/control points use in a polynomial spline in homogeneous coordinates, then the $P_i$ are the poles/control points in the corresponding rational spline, using the same "divide by w" projection from homogeneous coordinates to regular coordinates. Note that the denominator of the rational function is weight coefficient of the homogeneous slot in the original polynomial spline.

$$\begin{aligned}\hat{c}(u) &= \sum_{i=0}^{n-1} b_i(\mathrm{u})\hat{P}_i \\ &= \sum_{i=0}^{n-1} b_i(\mathrm{u})(w_i P_i, w_i) \\ &= \sum_{i=0}^{n-1} w_i b_i(\mathrm{u})(P_i, 1) \\ c(u) &= \sum_{i=0}^{n-1} w_i b_i(\mathrm{u}) P_i \Big/ \sum_{i=0}^{n-1} w_i b_i(\mathrm{u})\end{aligned}$$

*If isRational is TRUE, the control points of the spline are in homogeneous coordinates, each point having a weight.*

### 7.6.6 Interface: PolynomialSpline

#### 7.6.6.1 Semantics

A polynomial spline is an interpolation of datapoints, i.e. a polynomial curve passing through data points. Construction of such a spline depends on the constraints: which may include:

- on values or derivatives of the spline at the data points
- on the continuity of various derivatives at chosen points
- degree of the polynomial in use.

*An "nth degree" polynomial spline shall be defined for each direct position offset, piecewise between knot parameter values, as an n-degree polynomial, with up to $C^{n-1}$ continuity at the control points where the defining polynomial may change.*

*This level of continuity is controlled by the attribute numDerivativesInterior, which shall default to (degree-1).*

*Parameters may include directions for as many as degree – 2 derivatives of the polynomial at the start and end point of the segment.*

Line is a 1st degree polynomial spline. For line strings viewed as splines, it can be represented as almost any type, Bézier, B-Spline or Polynomial, since with degree=1, the formulae for each of these types collapse to a line string. Line strings have simple continuity at the controlPoints ($C^0$), but does not require derivative information (degree – 2 = –1).


© ISO 2013 – All rights reserved

The process for setting up the polynomial for each offset for a set of direct positions is solving a set of equations for the coefficients of the polynomial between each pair of sequential knots (note that for polynomial splines, all knots are degree 1). :

$$\text{knots: } k_i = [u_i, 1],$$
$$\text{dataPoints: } \vec{\mathrm{P}}_i = (\mathrm{x}_{i,0}, \mathrm{x}_{i,1}, \mathrm{x}_{i,2} \ldots),$$
$$c(u) = (c_o(u), c_1(u), c_2(u), \ldots),$$
$$c_j \text{ is a polynomtial on } [u_i, u_{i+1}]$$
$$c_j(u_i) = x_{i,j}$$
$$\frac{d}{du} c_j^-(u_i) = \frac{d}{du} c_j^+(u_i)$$

NOTE The major difference between the polynomial splines, the b-splines (basis splines) and Bézier splines is that polynomial splines pass through their control points, making the control point and sample point array identical. Polynomial splines are essentially calculated by brute force (albeit sometimes "organized" brute force) from the knots, the control points, and other parameters. Because of this, they do not generalize to surface splines, except in limited cases.

#### 7.6.6.2 Attribute: vectorAtStart: Vector [*] vectorAtEnd: Vector [*]

The attribute "vectorAtStart" shall be the values used for the initial derivative (up to degree – 2) used for interpolation in this PolynomialSpline at the start point of the spline. The attribute "vectorAtEnd" shall be the values used for the final derivative (up to degree – 2) used for interpolation in this PolynomialSpline at the start point of the spline.

$$n = knot.length - 1$$
$$vectorAtStart[0] = \dot{c}(\mathrm{u}_0),$$
$$vectorAtStart[1] = \ddot{c}(\mathrm{u}_0),$$
$$vectorAtStart[2] = \dddot{c}(\mathrm{u}_0) \ldots$$
$$vectorAtEnd[0] = \dot{c}(\mathrm{u}_n),$$
$$vectorAtEnd[1] = \ddot{c}(\mathrm{u}_n),$$
$$vectorAtEnd[2] = \dddot{c}(\mathrm{u}_n) \ldots$$

```
PolynomialSpline::vectorAtStart:Vector[*]
     {size ≤ degree - 2}
PolynomialSpline::vectorAtEnd:Vector[*]
     {size ≤ degree - 2}
```

### 7.6.7 Interface: CubicSpline

Cubic splines are similar to line strings in that they are a sequence of segments each with its own defining cubic polynomial splines. The usual constraints are:

© ISO 2013 – All rights reserved

$$\vec{c}(u_i) = \vec{P}_i$$
$$\dot{\vec{c}}^{-}(u_i) = \dot{\vec{c}}^{+}(u_i) \text{ for i=1,2...,n-2}$$
$$\dot{\vec{c}}(u_0) = vectorAtStart[0]$$
$$\dot{\vec{c}}(u_{n-1}) = vectorAtEnd[0]$$

A cubic spline uses the datapoints and a set of derivative parameters to define a piecewise third degree polynomial interpolation in each coordinate dimension. Because the dimensions are handled separately, homogeneous coordinates shall not be used in this curve type. This is true for any fitted curve regardless of polynomial degree. Unlike line-strings, the parameterization by arc length is not necessarily still a polynomial. Splines have two parameterizations that are used in this International Standard, the defining one (constructive parameter) and the one that has been parameterized by arc length to satisfy the requirements in Curve. In a CubicSpline, the constructive one is a set of cubic polynomials, on for each dimension offset in the DirectPosition used.

The function describing the curve must be $C^2$, that is, have a continuous first and second derivative at all points, and pass through the controlPoints in the order given. Between any two consecutive control points, the curve segment is defined by cubic polynomials, one for each offset in the coordinates. At each control point, the polynomial changes in such a manner that the first and second derivative vectors are the same from either side. The control parameters record must contain vectorAtStart, and vectorAtEnd which are the tangent vectors at controlPoint [1] and controlPoint [n] where n = controlPoint.count. Since the constructive parameterization is not the arc-length one, these tangent vectors may well not be unit vectors. The length of the vectors affects the shape of the curve.

The restriction on “vectorAtStart” and “vectorAtEnd” reduce these sequences to a single tangent vector each.

```
CubicSpline.vectorAtStart:Vector \\ "degree - 2" is 1
CubicSpline.vectorAtEnd:Vector \\ "degree - 2" is 1
```

NOTE The actual implementation of the cubic polynomials varies, but the curve so generated is guaranteed to be unique. The references [2], [10], [12], [18], and [19] in the bibliography contain examples of implementations.

The interpolation mechanism for a CubicSpline is “cubicSpline”.

```
CubicSpline::interpolation:InterpolationMethod="cubicSpline"
```

The degree for a CubicSpline is “3”.

```
CubicSpline□degree:Integer="3"
```

### 7.6.8 Interface: Bezier

The Bezier are approximating splines that use Bézier or Bernstein polynomials as a partition of unity, see Annex B.3.1. An n+1 long control point array will create a polynomial curve of degree n that defines the entire segment. These curves are defined in terms of the set of basis functions given by:

$$J_{n,i}(u) = \binom{n}{i} u^i (1-u)^{n-i} \text{ where } \binom{n}{i} = \frac{n!}{i!(n-i)!} \text{ for i = 0, 1, 2 … n}$$


© ISO 2013 – All rights reserved

These functions are a "Partition on Unity" which can be seen by using the Binomial theorem for degree n on $(x+y)^n$ where $x=u$ and $y=(1-u)$:

$$1=1^n=(u+(1-u))^n=\sum_{i=0}^{n}\binom{n}{i}u^i(1-u)^{n-i}=\sum_{i=0}^{n}J_{n,i}(u)$$

The only knots for a Bézier are "0" and "1" and they are multiplicity "n-1." The set of "n+1" control points $\vec{P}_0$, $\vec{P}_1$... $\vec{P}_n$ shall determine a curve segment given by:

$$\vec{c}(u)=\sum_{i=0}^{n}J_{n,i}(u)\vec{P}_i \quad \text{for } u\in[0,1].$$

The sample points of this segment are the values of the curve defined at the maximum of each of the polynomials (i/n):

$$\forall\ i\in\{0,\ 1,\ 2\ \ldots\ n\}:\vec{S}_i=\vec{c}\left(\frac{i}{n}\right)$$

The only control points that the curve is force to go through are the first and the last one in the array.

NOTE For n = 1, the two weight functions are as follows:

$$J_{1,0}(t)=\binom{1}{0}t^0(1-t)^1=(1-t)\quad\text{and}\quad J_{1,1}(t)=\binom{1}{1}t^1(1-t)^{1-1}=t$$

Given $P_0$ and $P_1$, the curve segment becomes:

$$\vec{c}(t)=(1-t)P_0+tP_1 \text{ for t} \in [0,1]$$

In other words, for $n = 1$, the Bézier polynomial is geometrically equivalent to a simple line segment.

If C is a Bezier with a uniform knot sequence, $c.knotType="piecewiseBezier"$ and of degree n, then the controlPoint sequence shall have length one greater than some integer multiple n; that is, there will exist an positive integer s such that:

$$\exists s\in\mathbb{Z}^+\ \ni\ c.controlPoint.length=s*c.degree+1$$

Further, for each subsequence of control points starting at an index of a multiple of n, of length n+1, the curve will be a Bézier spline of degree n starting at the first point in that subsequence and ending at the last point, with that subsequence of n+ 1 point as poles for that part of the curve. This curve will be continuous in all cases since the last pole of each Bézier curve is the first pole of the next. It will be $C^1$ (continuous first derivative) if the difference vector of the last two poles of one sequence is equal to the difference vector of the first two of the next sequence, i.e. if

$$[\overrightarrow{P}_{i*n}-\vec{P}_{i*n-1}=\vec{P}_{i*n+1}-\vec{P}_{i*n}]\Rightarrow[c' \text{ is contiuous at } \vec{P}_{i*n}]$$

© ISO 2013 – All rights reserved

### 7.6.9 Interface: BSplineCurve

#### 7.6.9.1 Semantics

A B-spline curve (Figure 24) is a piecewise parametric polynomial or rational curve described in terms of control points and basis functions. If the control points are not homogeneous form or they are but the weights are all equal to one another, then it a piecewise polynomial function. Otherwise, then it is a rational function spline. A B-spline curve is a piecewise Bézier curve if it is quasi-uniform except that the interior knots have multiplicity "degree" rather than having multiplicity one. In this subtype the knot spacing shall be 1.0, starting at 0.0. A piecewise Bézier curve that has only two knots, 0.0, and 1.0, each of multiplicity (degree+1), is equivalent to a simple Bézier curve.

#### 7.6.9.2 curveForm

The attribute "curveForm" is used to identify particular types of curve which this spline is being used to approximate. It is for information only, used to capture the original intention. If no such approximation is intended, then the value of this attribute is NULL.

```
BSplineCurve::curveForm:SplineCurveForm
```

#### 7.6.9.3 knotSpec

The attribute "knotSpec" gives the type of knot distribution used in defining this spline. This is for information only and is set according to the different construction-functions.

```
BSplineCurve::knotSpec[0,1]:KnotType
```

#### 7.6.9.4 BSplineCurve (constructor)

The class constructor "BSplineCurve" takes the pertinent information described in the attributes above and constructs a B-spline curve. If the knotSpec is not present, then the knotType is uniform and the knots are evenly spaced, and except for the first and last have multiplicity = 1. At the ends the knots are of multiplicity = degree+1. If the knotType is uniform they need not be specified.

```
BSplineCurve::BSplineCurve(deg:Integer, pts:PointArray,
k[0,1]:Sequence<Knot>, ks[0,1]:KnotType):BSplineCurve
```

NOTE If the B-spline curve is uniform and degree = 1, the B-spline is equivalent to a polyline (Line). If the knotType is "piecewiseBezier", then the knots are defaulted so that they are evenly spaced, and except for the first and last have multiplicity equal to degree. At the ends the knots are of multiplicity = degree+1.

## 8 Interpolations for Surfaces

### 8.1 Requirements class: Gridded surfaces

### 8.1.1 Interface: ParametricCurveSurface

#### 8.1.1.1 Semantics

The parametric curve surfaces are continuous families of curves, given by a constructive function of the form:

$$S(u,v):[a,b]\otimes[c,d]\rightarrow DirectPosition$$


© ISO 2013 – All rights reserved

A one-parameter family of curves can be defined by fixing the value of either parameter; fixing $v$ gives a set of curves in $u$, fixing $u$ gives a set of curves in $v$.

$$c_v(u) = c_u(v) = S(u,v)$$

The functions on ParametricCurveSurface (Figure 25) shall expose these two families of curves. The first gives us the "horizontal" cross sections $c_v(u)$, the later the "vertical" cross sections $c_u(v)$. The terms "horizontal" and "vertical" refer to the parameter space and need not be either horizontal or vertical in the coordinate reference system. Table 1 lists some possible pairs of types for these surface curves (other representations of these same surfaces are possible). The two partial derivatives of the surface parameterization, $\vec{i}$ and $\vec{j}$ are given by:

$$\vec{i} \equiv \frac{\partial}{\partial u} S(u,v) = \frac{d}{du} c_v(u)$$

and

$$\vec{j} \equiv \frac{\partial}{\partial v} S(u,v) = \frac{d}{dv} c_u(v)$$

The default upNormal (6.3.15.10) for the surface is the normalized vector cross product of these two curve derivatives when they are both non-zero: $\vec{n} = (\vec{i} \times \vec{j}) / \|\vec{i} \times \vec{j}\|$

If the coordinate reference system is 2D, then the vector $\vec{n}$ extends the local coordinate system by supplying an "upward" elevation vector. In this case the vector basis $(\vec{i}, \vec{j})$ must be a right hand system; that is to say, the counter-clockwise oriented angle from *i* to *j* must be less than 180°. This gives a right-handed "moving frame" for each curve of local coordinate axes given by $< \vec{i}, \vec{j} >$. A moving frame is defined to be a continuous function from the geometric object to a basis for the local tangent space of that object. For the section curves, this is the derivative of the curve, the local tangent. For a parametric curve surface, this is a local pair of tangents.

© ISO 2013 – All rights reserved

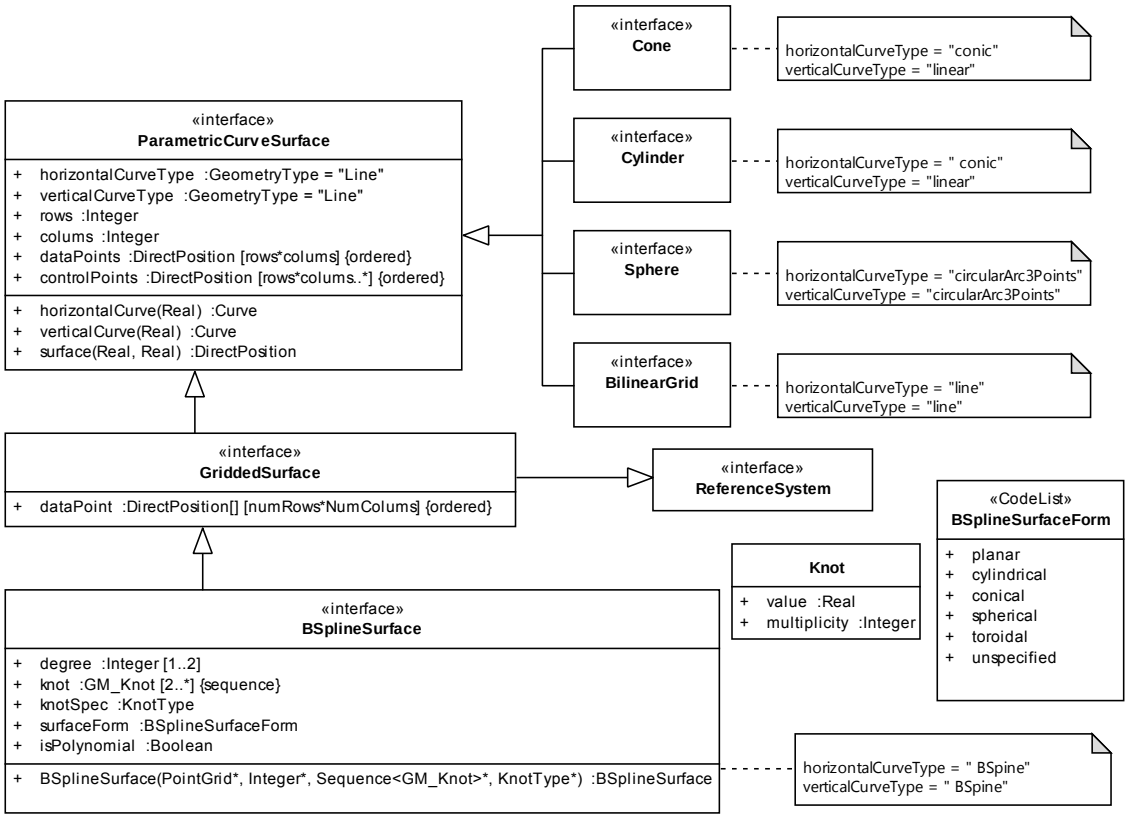



**Figure 25 — ParametricCurveSurface and its subtypes**

NOTE The existence of a viable moving frame is the definition of "orientable" manifold. This is why the existence of a continuous upNormal implies that the surface is orientable. Non-orientable surfaces, such as the Möbius band and Klein bottle are counter-intuitive. Clause 6.3.15.10 forbids their use in application schemas conforming to this International Standard. Klein bottles cannot even be constructed in 3D space, but require 4D space for non-singular representations.

**Table 1 — Examples of parametric curve representations**

| Surface type | Horizontal curve type | Vertical curve type |
|---|---|---|
| Cylinder | Circle, constant radii | Line Segment |
| Cone | Circle, increasing radii from the apex point | Line Segment |
| Sphere | Circle of constant latitude | Circle of constant longitude |
| BilinearGrid | Line string | Line string |
| BicubicGrid | Cubic spline | Cubic spline |

The rows from the parameter grid are associated to dataPoints for the horizontal surface curves; the columns are associated to dataPoints for the vertical surface curves. The working assumption is that for a pair of parametric coordinates $(u,v)$ that the horizontal curves for each integer offset are calculated and evaluated at "$u$". This defines a sequence of control points:

$$\{S(u,v_n)|\} = \{c_{v_n}(u) : u = k_{0,n}, \ldots k_{columns-1,n}\}$$

From this sequence, a vertical curve is calculated for "$u$", and evaluated at "$v$". The order of calculation (horizontal-vertical versus vertical-horizontal) does not normally make a difference. Where it does, the horizontal-vertical order should be used.

NOTE A common case of a parametric surface S is a 2D spline. In this case the weight functions for each parameter make order of calculation unimportant:


© ISO 2013 – All rights reserved

$$S(u,v) = \sum_{i=0}^{rows-1} \sum_{j=0}^{colums-1} w_i^u(u) w_j^v(v) \vec{P}_{i,j}$$

where $\vec{P}_{i,j}$ is the control point in the $i^{th}$ row and $j^{th}$ column.

Logically, any pair of curve interpolation types can lead to a subtype of ParametricCureSurface.

#### 8.1.1.2 Attribute: ParametricCurveSurface.horizontalCurveType ParametricCurveSurface.verticalCurveType

The attribute "horizontalCurveType" indicates the type of section curves used to traverse the surface "horizontally" which fix the parameter "*v*" and uses the parameter "*u*", $S.horizontalCurve(v)(u) = c_v(u)$.

```
ParametricCurveSurface::horizontalCurveType:GeometryType
```

*The GeometryType returned by horizontalCurveType shall be in the local code list GeometryType and shall be a subtype of Curve.*

The attribute "verticalCurveType" indicates the type of surface curves used to traverse the surface vertically which fix the parameter "*u*" and uses the parameter "*v"*, $S.verticalCurve(u)(v) = c_u(v)$.

*The GeometryType returned by verticalCurveType shall be in the local code list GeometryType and shall be a subtype of Curve.*

```
ParametricCurveSurface::verticalCurveType: GeometryType
```

#### 8.1.1.3 Attribute: ParametricCurveSurface.rows

The derived attribute "rows" gives the number of rows in the parameter grid.

```
ParametricCurveSurface::rows:Integer
```

#### 8.1.1.4 Attribute: ParametricCurveSurface.columns

The derived attribute "columns" gives the number of columns in the parameter grid.

```
ParametricCurveSurface::columns:Integer
```

#### 8.1.1.5 Attribute: ParametricCurveSurface.dataPoints[]

The dataPoints are the functional images of the knots in the parameter grid.

$$knots = (u_i, v_j),$$
$$dataPoint(i,j) = S(u_i, v_j) = c_{u_i}(v_j) = c_{v_j}(u_i)$$

© ISO 2013 – All rights reserved

#### 8.1.1.6 Attribute: ParametricCurveSurface.comtrolPoints[]

If either the horizontal or vertical curves require control points in their definition, they may be accessed via the control point array. There will be (rows × columns) points in the control point array, stored in row-major form.

#### 8.1.1.7 Operation: ParametricCurveSurface.horizontalCurve: Curve

The operation "horizontalCurve" constructs a curve that traverses the surface horizontally with respect to the parameter "*s*". This curve holds the parameter "*t*" constant.

```
ParametricCurveSurface::horizontalCurve(v:Real):Curve
```

NOTE The Curve returned by this function or by the corresponding vertical curve function, are normally not part of any GeometryComplex to which this surface is included. These are, in general, calculated transient values. The exceptions to this may occur at the extremes of the parameter space. The boundaries of the parameter space support for the surface map normally to the boundaries of the target surfaces.

#### 8.1.1.8 Operation: verticalCurve: Curve

The operation "verticalCurve" constructs a curve that traverses the surface vertically with respect to the parameter "*t*". This curve holds the parameter "*s*" constant.

```
ParametricCurveSurface::verticalCurve(u:Real):Curve
```

#### 8.1.1.9 Operation: surface

The operation "surface" traverses the surface both vertically and horizontally.

```
ParametricCurveSurface::surface(u:Real,v:Real):
     DirectPosition
```

### 8.1.2 Interface: GriddedSurface (Tensor Spline)

The GriddedSurface (Figure 25) is a ParametricCurveSurface defined from a rectangular grid in the parameter space, in which the interior of the domain is locally one to one, i.e:

$$S : [u_0, u_{columns-1}] \otimes [v_0, v_{rows-1}] \rightarrow \text{DirectPosition}$$
$$\forall (\mathrm{u}_i, \mathrm{v}_j), (\mathrm{u}_n, v_m) \in (u_0, u_{columns-1}) \otimes (v_0, v_{rows-1}) :$$
$$\mathrm{S}(\mathrm{u}_i, \mathrm{v}_j) = \mathrm{S}(\mathrm{u}_n, v_m) \Leftrightarrow (\mathrm{u}_i, \mathrm{v}_j) = (\mathrm{u}_n, v_m)$$

This limitation allows the parameter space of the surface to be used as a CRS for the direct position on the surface. The metric can be derived from the space containing the surface that implies that the length of a cure on the surface is the length of that curve in the CRS of the surface.

Example If the surface is a Figure of The Earth ellipsoid, and the horizontal curves are of constant longitude (the parameter is then latitude) and the vertical cures are of constant latitude (the parameter is then longitude); the parameter may maps ($\phi$, $\lambda$) to positions on the ellipsoid.

Example If the surface is the range of an image, the parameter surface is essentially a gridded coverage, and the parameter map maps the image onto the surface of the Figure of the Earth being used.

 © ISO 2013 – All rights reserved

### 8.1.3 Interface: BilinearGrid

A BilinearGrid is a ParametricCurveSurface that uses line strings as the horizontal and vertical curves.

Example On a 2 by 2 grid, the equations are quite symmetric (using integer knots, the data points at the knots are the data points for that particular crossing, (integer, integer) coordinate):

$$\{S(0,0)=\vec{P}_{0,0}, S(1,0)=\vec{P}_{1,0}, S(0,1)=\vec{P}_{0,1}, S(1,1)=\vec{P}_{1,1}\}:$$
$$S(u,v)=\vec{P}_{u,v}=(1-u)(1-v)\vec{P}_{0,0}+(1-u)v\vec{P}_{0,1}+u(1-v)\vec{P}_{1,0}+uv\vec{P}_{1,1}$$

If the four points are coplanar, the section of the grid is a polygon.

NOTE This is not a polygonal surface, since each of the grid squares is a ruled surface, and not necessarily planar.

Another internal representation is the use of real valued polynomials (one for each coordinate dimension, as defined by the CRS.dimension $d$ – clause 6.2.2.3), on 2 variable (here we will use $(u,v)\in[a,b]\otimes[c,d]$)

$$P(\mathrm{u},\mathrm{v})=\begin{bmatrix} p_0(\mathrm{u},\mathrm{v}) \\ p_1(\mathrm{u},\mathrm{v}) \\ \dots \\ p_{d-1}(\mathrm{u},\mathrm{v}) \end{bmatrix}$$

$$p_i(u,v)=\begin{bmatrix}1 & u & \dots & u^n\end{bmatrix}\begin{bmatrix} a^i_{00} & a^i_{01} & \dots & a^i_{0m} \\ a^i_{10} & a^i_{11} & \dots & a^i_{1m} \\ \dots & \dots & \dots & \dots \\ a^i_{n0} & & & a^i_{nm}\end{bmatrix}\begin{bmatrix}1 \\ v \\ \dots \\ v^m\end{bmatrix}=\sum_{r=o,c=o}^{n,m} a^i_{rc}u^r v^c$$

## 8.2 Requirements class: Polygon

### 8.2.1 Interface: Polygon

#### 8.2.1.1 Semantics

A Polygon (Figure 26) is a surface patch that is defined by a set of boundary curves and an underlying surface to which these curves adhere. The default logic has been that a polygon was "planer" and the underlying coordinate system was therefore Euclidean. In general, any topological 2D surface with a local coordinate system will work, e.g. see 8.1.2 Interface: GriddedSurface.

The default is that the curves are coplanar, or lie in the same geodesic surface (Figure of the Earth), and the polygon uses planar interpolation in its interior.

Note: If the spanning surface is a GriddedSurface, then the boundary curves can use the parameteric coordinates of the surface in lieu of a more standard CRS.

#### 8.2.1.2 Attribute: Polygon: boundary: Curve[1...*]

The attribute "boundary" stores the surface boundary cures that are the boundary of this Polygon.

```
Polygon::boundary:Curve[1…*]
```

© ISO 2013 – All rights reserved

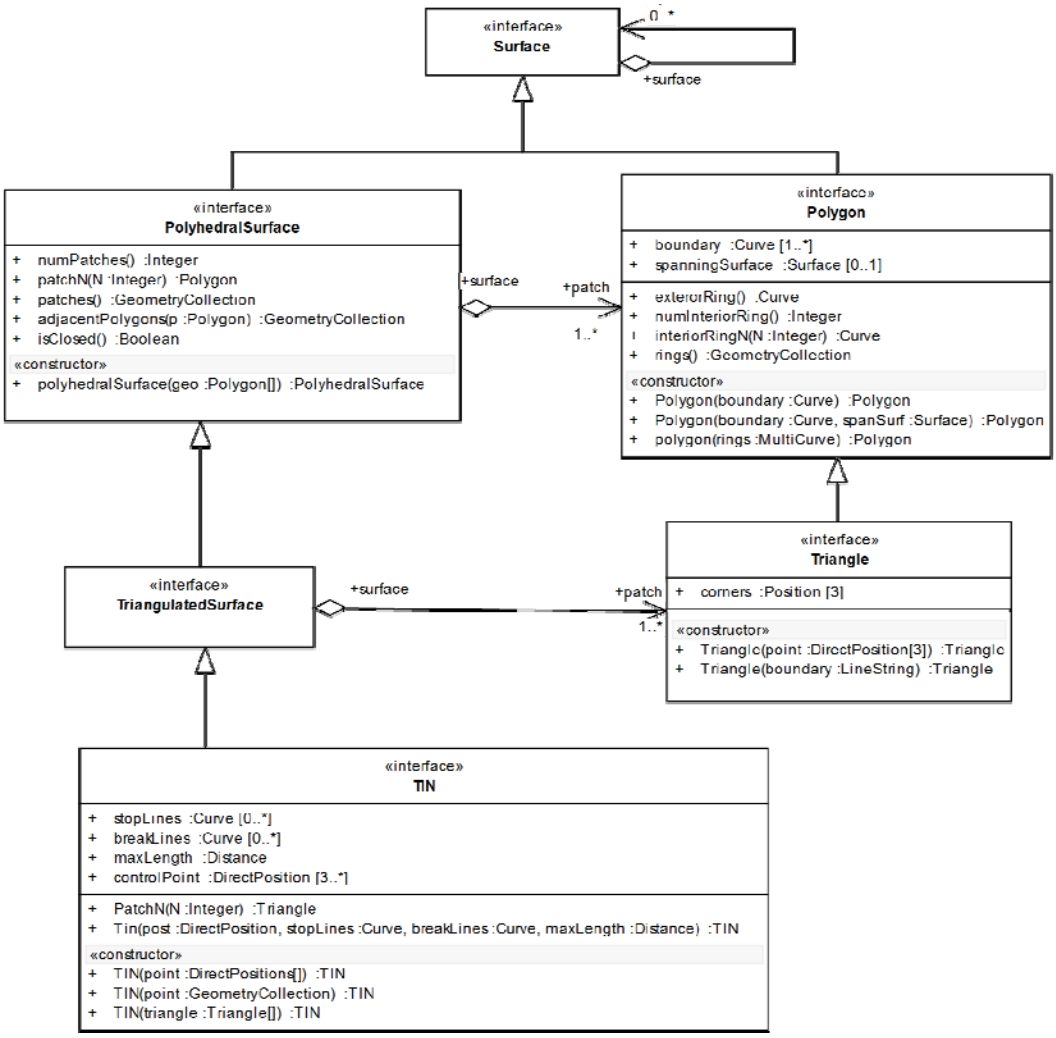



**Figure 26— Polygonal surface**

### 8.2.1.3 Association Role: spanningSurface [0.1]

The optional spanning surface provides a mechanism for spanning the interior of the polygon.

```
Polygon::spanningSurface[0,1]:Surface
```

NOTE The spanning surface should have no boundary components that intersect the boundary of the polygon, and there should be no ambiguity as to which portion of the surface is described by the bounding curves for the polygon. A common spanning surface is an elevation model, which is not directly described in this International Standard, although Tins and gridded surfaces are often used in this role.

Any gridded surface can easily be used. In this case the CRS of the Polygon can ghe the local coordinates of the gridded surface parameter space.

### 8.2.1.4 Operation: Polygon (constructor)

This first variant of a constructor of Polygon creates a Polygon directly from a set of boundary curves which shall be defined using coplanar DirectPositions as controlPoints.

```
Polygon::Polygon(boundary:Curve[1..*]):Polygon
```


© ISO 2013 – All rights reserved

NOTE The meaning of exterior in the is consistent with the plane of the constructed planar polygon.

This second variant of a constructor of Polygon creates a Polygon lying on a spanning surface. There is no restriction of the types of interpolation used by the composite curves used in the surface boundary, but they must all be lie on the "spanningSurface" for the process to succeed.

```
Polygon(boundary:Curve[1..*], spanSurf:Surface):Polygon
```

NOTE It is important that the boundary components be oriented properly for this to work. It is often the case that in bounded manifolds, such as the sphere, there is an ambiguity unless the orientation is properly used.

If the spanning surface is a gridded surface (Tensor Surface) then the CRS of the curves may be the parameter space of the surface.

### 8.2.2 Interface: PolyhedralSurface

#### 8.2.2.1 Semantics

A PolyhedralSurface (Figure 26) is a Surface composed of polygon surfaces (Polygon) connected along their common boundary curves. This differs from Surface only in the restriction on the types of surface patches acceptable.

#### 8.2.2.2 PolyhedralSurface (constructor)

The constructor for a PolyhedralSurface takes the facet Polygons and creates the necessary aggregate surface.

```
PolyhedralSurface::PolyhedralSurace(tiles[1..n]:Polygon:
     PolyhedralSurface
```

#### 8.2.2.3 patch

The association role "patch" associates this surface with its individual facet polygons. It shall be non-empty.

```
PolyhedralSurface::patch[1,n]:Reference<Polygon>
```

### 8.2.3 Interface: Triangle

A Triangle is a planar Polygon defined by three corners; that is, a Triangle would be the result of a constructor of the form:

```
Polygon(Line(<P1, P2, P3, P1>))
```

where P1, P2, and P3 are three DirectPositions. Triangles have no holes. Triangle shall be used to construct TriangulatedSurfaces.

NOTE The points in a triangle can be located in terms of their corner points by defining a set of barycentric coordinates, three nonnegative numbers $c_1$, $c_2$, and $c_3$ such that $c_1 + c_2 + c_3 = 1.0$. Then, each point $P$ in the triangle can be expressed for some set of barycentric coordinates as:

$$P = c_1 P_1 + c_2 P_2 + c_3 P_3$$

© ISO 2013 – All rights reserved

### 8.2.4 Interface: TriangulatedSurface

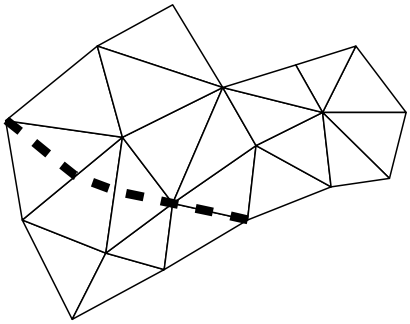
A TriangulatedSurface (Figure 21) is a PolyhedralSurface that is composed only of triangles (Triangle). There is no restriction on how the triangulation is derived.

### 8.2.5 Interface: Tin

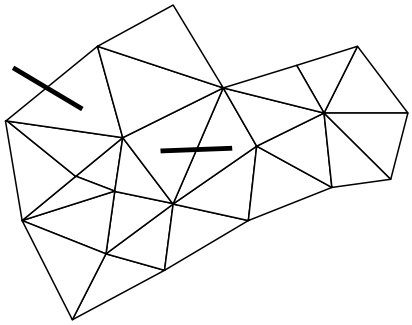
#### 8.2.5.1 Semantics

ED: The description of TIN construction will be updated to conform to other ISO and JTC1 standards.

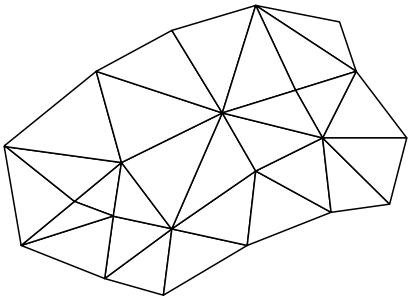
A Tin (Figure 21) is a TriangulatedSurface that uses the Delaunay algorithm or a similar algorithm complemented with consideration for breaklines, stoplines and maximum length of triangle sides (Figure 22). These networks satisfy the Delaunay criterion away from the modifications: For each triangle in the network, the circle passing through its vertexes does not contain, in its interior, the vertex of any other triangle.



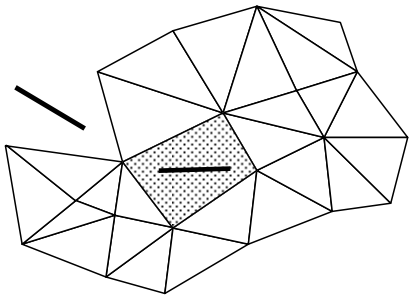
a) First triangulation (Delauny) not using breakline



b) Additional stoplines



c) Retriangulation using breakline



d) Retriangulation using stoplines with resulting holes and boundary changes

**Figure 27 — TIN construction**

#### 8.2.5.2 stopLines

Stoplines are lines where the local continuity or regularity of the surface is questionable. In the area of these pathologies, triangles intersecting a stopline shall be removed from the TIN surface, leaving holes in the surface. If coincidence occurs on surface boundary triangles, the result shall be a change of the surface boundary. The attribute “stopLines” contains all these pathological segments as a set of line strings.

```
Tin::stopLines:Set<Line>
```


© ISO 2013 – All rights reserved

#### 8.2.5.3 breakLines

Breaklines are lines of a critical nature to the shape of the surface, representing local ridges, or depressions (such as drainage lines) in the surface. As such their constituent segments must be included in the TIN even if doing so violates the Delaunay criterion. The attribute "breakLines" contains these critical segments as a set of line strings.

```
Tin::breakLines:Set<Line>
```

#### 8.2.5.4 maxLength

Areas of the surface where the data is not sufficiently dense to assure reasonable calculations shall be removed by adding a retention criterion for triangles based on the length of their sides. For any triangle sides exceeding maximum length, the adjacent triangles to that triangle side shall be removed from the surface.

```
Tin::maxLength:Distance
```

#### 8.2.5.5 controlPoint

The corners of the triangles in the TIN are often referred to as posts. The attribute "controlPoint" shall contain a set of the DirectPositions used as posts for this TIN. Since each TIN contains triangles, there must be at least three posts. The order in which these points are given does not affect the surface that is represented. Application schemas may add information based on the ordering of the control points to facilitate the reconstruction of the TIN from the controlPoints.

```
Tin::controlPoint[3..n]:DirectPosition
```

NOTE The control points of a TIN are often called "posts".

#### 8.2.5.6 Tin (constructor)

The constructor for a restricted Delaunay network requires the triangle corners (posts), breaklines, stoplines, and maximum length of a triangle side.

```
Tin::Tin(post:Set<DirectPosition>, stopLines:Set<Line>,
breakLines:Set<Line>, maxLength:Number):Tin
```

### 8.3 Requirements class: Conic Surface

#### 8.3.1 Interface: Sphere

A Sphere is a GriddedSurface given as a family of circles whose positions vary linearly along the axis of the sphere, and whose radius varies in proportion to the cosine function of the central angle. The horizontal circles resemble lines of constant latitude, and the vertical arcs resemble lines of constant longitude.

Note If the control points are sorted in terms of increasing longitude, and increasing latitude, the upNormal of a sphere is the outward normal.

Example If we take a gridded set of latitudes and longitudes in degrees, $(\varphi,\lambda)$, such as

© ISO 2013 – All rights reserved

| | | | | |
|---|---|---|---|---|
| (–90, –180) | (–90, –90) | (–90,0) | (–90, 90) | (–90, 180) |
| (–45, –180) | (–45, –90) | (–45,0) | (–45, 90) | (–45, 180) |
| (0, –180) | (0, –90) | (0,0) | (0, 90) | (0, 180) |
| (45, –180) | (45, –90) | (45,0) | (45, –90) | (45, 180) |
| (90, –180) | (90, –90) | (90,0) | (90, –90) | (90, 180) |

And map these points to 3D using the usual equations (where R is the radius of the required sphere).

$$x = \rho \cos\lambda \cos\varphi$$
$$y = \rho \cos\lambda \sin\varphi$$
$$z = \rho \sin\lambda$$

We have a sphere of radius $\rho$, centered at (0, 0), as a gridded surface. Notice that the entire first row and the entire last row of the control points map to a single point each in 3D Euclidean space, North and South poles respectively, and that each horizontal curve closes back on it self forming a geometric cycle. This gives us a metrically bounded (of finite size), topologically unbounded (not having a boundary, a cycle) surface.

### 8.3.2 Interface: Cone

A Cone is a GriddedSurface given as a family of conic sections whose controlPoints vary linearly.

NOTE A 5-point ellipse with all defining positions identical is a point. Thus, a truncated elliptical cone can be given as a 2 × 5 set of control points <<P1, P1, P1, P1, P1>, <P2, P3, P4, P5, P6>>. P1 is the apex of the cone. P2, P3, P4, P5, and P6 are any five distinct points around the base ellipse of the cone. If the horizontal curves are circles as opposed to ellipses; then a circular cone can be constructed using <<P1, P1, P1>, <P2, P3, P4>>.

### 8.3.3 Interface: Cylinder

A Cylinder is a GriddedSurface given as a family of circles whose positions vary along a set of parallel lines, keeping the cross sectional horizontal curves of a constant shape.

NOTE Given the same working assumptions as in the previous note, a Cylinder can be given by two circles, giving us control points of the form <<P1, P2, P3>, <P4, P5, P6>>.

## 8.4 Requirements class: Spline Surface

### 8.4.1 Interface: BSplineSurface

#### 8.4.1.1 Semantics

A B-spline surface is a rational or polynomial parametric surface that is represented by control points, basis functions and possibly weights. If the weights are all equal then the spline is piecewise polynomial. If they are not equal, then the spline is piecewise rational. If the Boolean “isPolynomial” is set to TRUE then the weights shall all be set to 1.

#### 8.4.1.2 degree

The attribute “degree” shall be the algebraic degree of the basis functions for the first and second parameter. If only one value is given, then the two degrees are equal.

```
BSplineSurface::degree[1,2]:Integer
```


© ISO 2013 – All rights reserved

#### 8.4.1.3 surfaceForm

The attribute “surfaceForm” is used to identify particular types of surface which this spline is being used to approximate. It is for information only, used to capture the original intention. If no such approximation is intended, then the value of this attribute is NULL.

```
BSplineSurface::surfaceForm:BSplineSurfaceForm
```

#### 8.4.1.4 knot

The attribute “knot” shall be two sequences of distinct knots used to define the B-spline basis functions for the two parameters. Recall that the knot data type holds information on knot multiplicity.

```
BSplineSurface::knot[2]:Sequence<Knot>
```

#### 8.4.1.5 knotSpec

The attribute “knotSpec” gives the type of knot distribution used in defining this spline. This is for information only and is set according to the different construction-functions.

```
BSplineSurface::knotSpec[0,1]:KnotType
```

#### 8.4.1.6 isPolynomial

The attribute “isPolynomial” is set to “True” if this is a polynomial spline.

```
BSplineSurface::isPolynomial:Boolean
```

#### 8.4.1.7 BSplineSurface (constructor)

The class constructor “BSplineSurface” takes the pertinent information described in the attributes above and constructs a B-spline surface. If the knotSpec is not present, then the knotType is uniform and the knots are evenly spaced, and, except for the first and last, have multiplicity = 1. At the ends the knots are of multiplicity = degree+1. If the knotType is uniform they need not be specified.

```
BSplineSurface::BSplineSurface(
pts:Sequence<PointArray>,
deg[1,2]:Integer,
k[0,2]:Sequence<Knot>,
ks[0,1]:KnotType):BSplineSurface
```

### 8.4.2 Interface: BSplineSurfaceForm

The code list “BSplineSurfaceForm” shall be used to indicate a particular geometric form represented by a BSplineSurface. The potential values are:

6. planar — a bounded portion of a plane represented by a B-spline surface of degree 1 in each parameter.
7. cylindrical — a bounded portion of a cylindrical surface represented by a B-spline surface.

© ISO 2013 – All rights reserved

8. conical — a bounded portion of the surface of a right circular cone represented by a B-spline surface.
9. spherical — a bounded portion of a sphere, or a complete sphere represented by a B-spline surface.
10. toroidal — a torus or a portion of a torus represented by a B-spline surface.
11. unspecified — no particular surface is specified.

```
BSplineSurfaceForm::
planar
cylindrical
conical
spherical
toroidal
unspecified
```

# 9 Interpolations for Solids

## 9.1 Requirements class: Boundary Representation

### 9.1.1 Semantics

The simplest representation of a solid in a 3D space is by specification of its boundary surfaces. By the Jordan separation theorem, a closed simple 3D surface divides a $\mathbb{E}^3$ space into 2 solid components, one finite and one infinite. In other coordinate systems, the normals to the surface point outward from the defined solid, and so there is no ambiguity in terms of which solid is being defined.

The root class surface (Figure 28) is sufficient to support boundary representations, where a solid, like a polygon, is represented by it boundary. This class can be support by an application only if at least one surface requirements class with realizable surfaces is also supported.

### 9.1.2 Interface: Solid

The interface Solid defined in Clause 6.3.17 is sufficient to support a boundary representation.

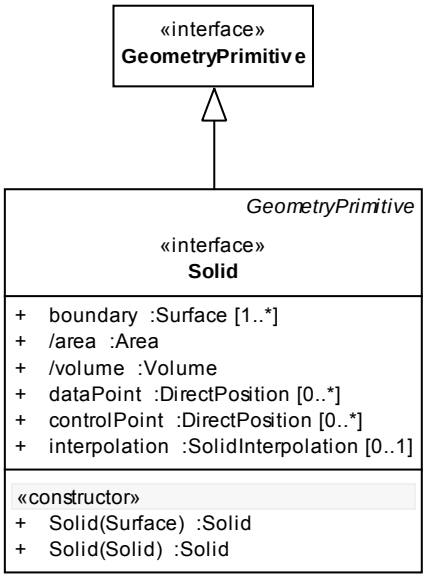



**Figure 28 — Solid boundary representation**


© ISO 2013 – All rights reserved

## 9.2 Requirements class: Gridded Solid

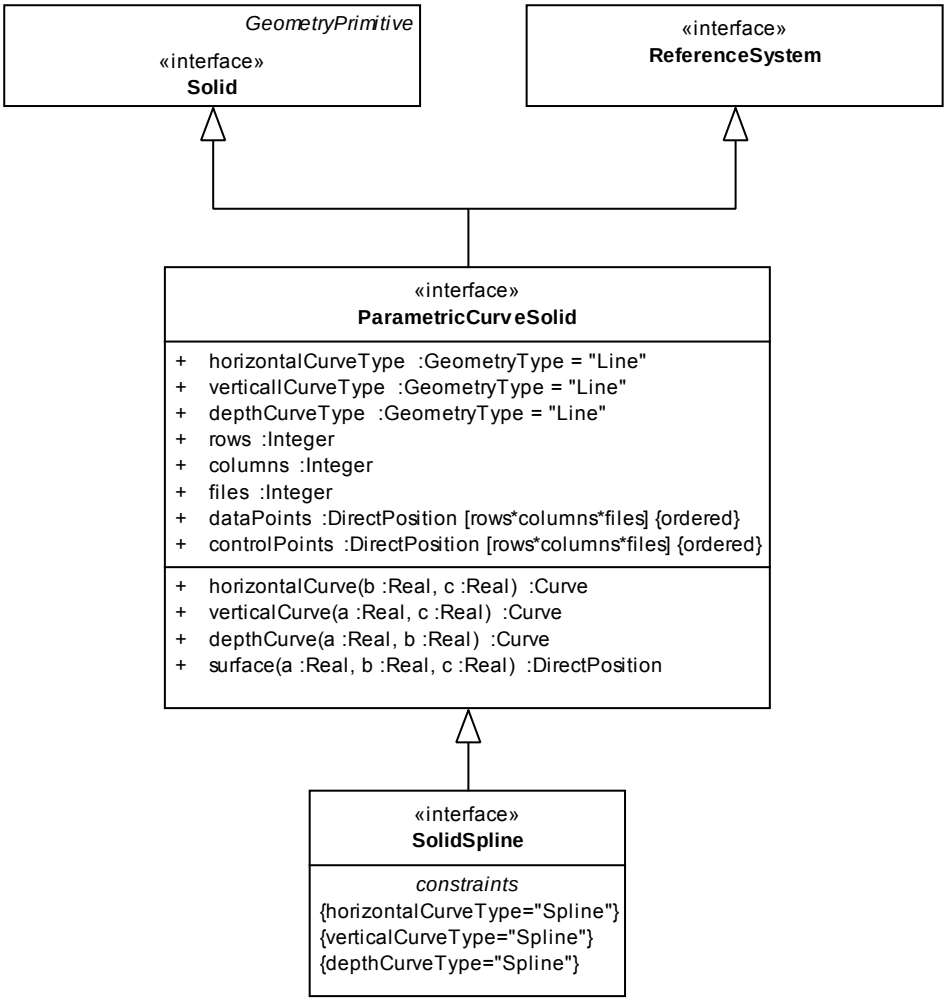



**Figure 29 — Parametric Solid**

### 9.2.1 Interface ParametricCurveSolid

#### 9.2.1.1 Semantics

This class does for solids what the interface ParametricCurveSurface (Clause 8.1.1) does for surfaces, it builds solids by bundling curves.

#### 9.2.1.2 Attribute: horizontalCurveType: GeometryType verticalCurveType: GeometryType depthCurveType: GeometryType

***The GeometryType returned by horizontalCurveType, verticalCurveType and depthCurveType shall be in the local code list GeometryType and shall be a subtype of Curve.***

```
ParametricCurveSolid::horizontalCurveType: GeometryType
ParametricCurveSolid::verticalCurveType: GeometryType
ParametricCurveSolid::depthCurveType: GeometryType
```

© ISO 2013 – All rights reserved

#### 9.2.1.3 Attribute:
**rows: Integer**
**columns: Integer**
**files: Integer**

The derived attribute "rows" gives the number of horizontal rows in the parameter grid. The derived attribute "columns" gives the number of vertical columns in the parameter grid. The derived attribute "files" gives the number of depth files in the parameter grid.

```
ParametricCurveSolid::rows:Integer
ParametricCurveSolid::columns:Integer
ParametricCurveSolid::files:Integer
```

#### 9.2.1.4 Attribute:
**dataPoint: DirectPosition[rows × columns × files]**
**controlPoint: DirectPosition[rows × columns × files]**

The dataPoints are the functional images of the knots in the 3D parameter grid.

$$knots = (u_i, v_j, t_k),$$
$$dataPo\,\text{int}(i,j,k) = S(u_i, v_j, t_k) = c_{u_i,t_k}(v_j) = c_{v_j,t_k}(u_i) = c_{u_i,v_j}(t_k)$$

If either the horizontal, vertical or depth curves require control points in their definition, they may be accessed via the control point array.

There will be (rows × columns × files) points in the control point array, stored in row-major form.

### 9.2.2 Interface SolidSpline

Solid splines are 3D parametric curve solids, where the various curves are splines (b-splines normally), and the equation for the solid interior is:

$$s_{p,q,r}(u,v,t) = \sum_{i=0}^{p}\sum_{j=0}^{q}\sum_{k=0}^{r} N_{i,p}(u)N_{j,q}(v)N_{k,r}(t)\overrightarrow{P_{i,j,k}}$$


© ISO 2013 – All rights reserved

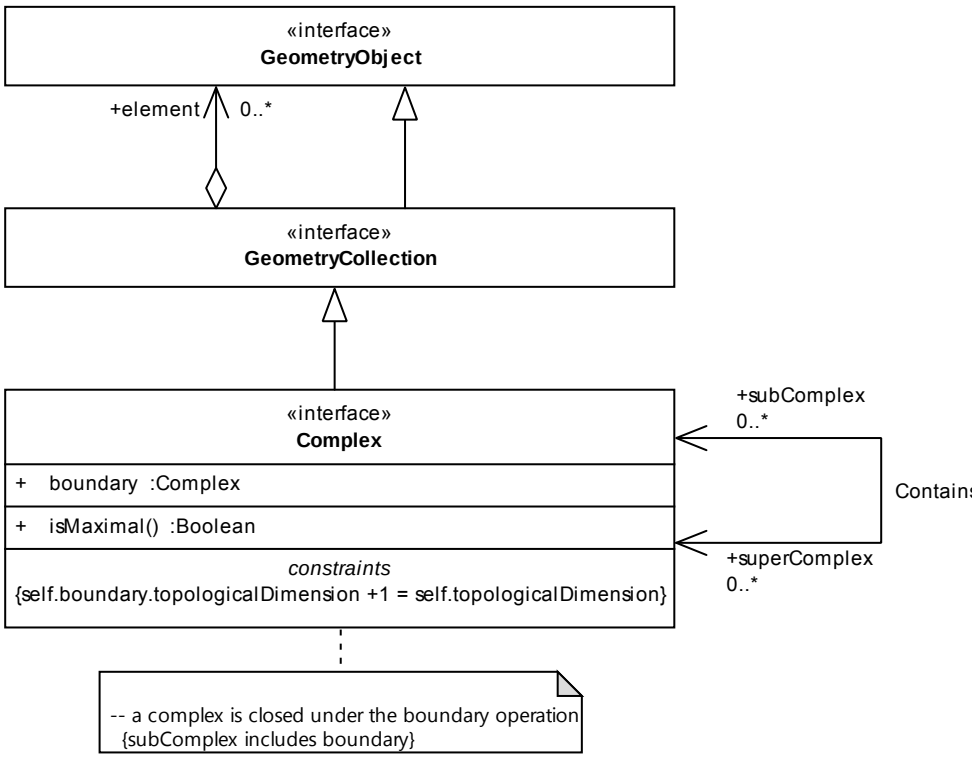



**Figure 30 — Geometric Complex**

## 10 Requirements class: Geometric complex

### 10.1 Semantics

A geometric complex (GeometryComplex) is a set of primitive geometric objects of the same topological dimension (in a common coordinate system) whose *interiors* are disjoint. Further, if a primitive is in a geometric complex (as a set of primitives), then there exists a set of primitives one dimension smaller in that complex whose point-wise union is the boundary of this first primitive.

A subcomplex of a complex is a subset of the primitives of that complex that is, in its own right, a geometric complex. A supercomplex of a complex is a superset of primitives that is also a complex. These definitions are essentially subset and superset with the added restriction that they must be a complex. A complex is maximal if it is a subcomplex of no larger complex instantiated.

The boundary of a geometric object in a geometric complex is a subcomplex of that complex. The simplest complex is a single point. The simplest 1-dimensional complex is a curve with its two end points. The simplest 2-dimensional complex is a surface with its boundary curve, and the curve's start and end points.

The underlying geometry of a complex is usually referred to as a "manifold". The structure of a complex organizes the geometry of the manifold into primitive elements, analogously to the way in which "charts" are organized by an "atlas" into a map of the world.

One way, but obviously not the only way, to generate a complex from a set of primitives is by beginning with those primitives and performing the following operations.

If two primitives overlap, then subdivide them, eliminating repetitions until there is no overlap.

© ISO 2013 – All rights reserved

Similarly, if a primitive is not simple, subdivide it where it intersects itself, eliminating repetitions until there is no overlap.

If a primitive is not a point, calculate its boundary as a collection of other primitives, using those already in the generating set if possible, and insert them into the complex.

Repeat step “a” through “c” until no new primitive is required.

Many systems have a concept of a universal face (for 2D) or universal solid (for 3D). This is valid only in the case where the underlying space of the complex is an unbounded Euclidean space. In this case, for 2D, the universal face is the surface in the GeometryComplex that has only interior boundary rings (its exterior one being the “point at infinity”). Analogously, in 3D, the universal solid is the one that has only interior boundary shells. In bounded manifolds, such as the sphere, there is no point at infinity, and all primitives are bounded. Without the Jordan Separation Theorem, all boundaries are essentially interior boundaries. In other unbounded manifolds, such as a hyperbolic surface, there may be more than one unbounded primitive. Since this International Standard does not directly address these sorts of unbounded manifolds, the cardinality of some elements may require relaxing if this International Standard were to be applied to such non-geographic manifolds. This International Standard does not special case either the universal face or solid, and the relationship between them and their boundaries are represented in the same manner as any other boundary relationship.

NOTE A maximal complex could reasonably be considered a strong aggregation of its primitives depending on the internal semantics of the application. For this reason, the mechanism for the containment of GeometryPrimitives in a GeometryComplex is left unspecified. If a strong aggregation is used for maximal complexes, then the containment association for subcomplexes may have to use the maximal complex as a namespace for the references to primitives within it. In any case, once a GeometryPrimitive is within a complex, or a GeometryComplex is a subcomplex of a maximal GeometryComplex, its boundary operation will not need to construct representative GeometryObjects, since by the definition of a complex, the objects needed to represent the boundary of the contained object will already exist, and only references to those objects are required by the GeometryObject::boundary operation. Remember that the containment of GeometryComplexes in one another is a subset-superset association, while the containment of GeometryPrimitives in a GeometryComplex is an element-set association.

## 10.2 Interface: Complex

### 10.2.1 Semantics

A geometry Complex (Figure 30) is a collection of geometrically disjoint, simple GeometryPrimitives. If a GeometryPrimitive (other than a Point) is in a particular GeometryComplex, then there exists a set of primitives of lower dimension in the same complex that form the boundary of this primitive.

NOTE A geometric complex can be thought of as a set in two distinct ways. First, it is a finite set of objects (via delegation to its elements member) and, second, it is an infinite set of point values as a subtype of geometric object. The dual use of delegation and subtyping is to disambiguate the two types of set interface. To determine if a GeometryPrimitive P is an element of a GeometryComplex C, call: C.element ().contains (P).

The “element” attribute allows GeometryComplex to inherit the behavior of Set<GeometryPrimitive> without confusing the same sort of behaviour inherited from TransfiniteSet<DirectPosition> inherited through GeometryObject.

Complexes shall be used in application schemas where the sharing of geometry is important, such as in the use of computational topology. In a complex, primitives may be aggregated many-to-many into composites for use as attributes of features. Examples of this are provided in the schemas in Annex D.

 © ISO 2013 – All rights reserved

### 10.2.2 isMaximal

The Boolean valued operation "isMaximal" shall return TRUE if and only if this GeometryComplex is maximal.

```
Complex::isMaximal():Boolean
```

### 10.2.3 Contains association

The association "Contains" instantiates the contains operation from Set<GeometryPrimitive> as an association.

```
GeometryComplex::subComplex[0..n]:GeometryComplex
GeometryComplex::superComplex[0..n]:GeometryComplex
```

### 10.2.4 Complex association

The association "Complex" is defined by the "contains" operation in GeometryObject that is inherited from TransfiniteSet<DirectPosition>.

```
Complex::element[1..n]:GeometryPrimitive
```

If a complex contains a GeometryPrimitive, then it must also contain the elements of its boundary.

```
Complex:
-- closed under the boundary operation
self→forAll(self→includesAll(boundary()))
```

## 11 Package: Topology interfaces

### 11.1 Semantics

The most productive use of topology is to accelerate computational geometry. The method by which this is accomplished is to associate explicitly feature instances and geometric object instances in a manner consistent with and derived from their implicit geometric relations (see D.3). In some cases, these associations are derived from a conceptual geometry that does not agree with the representation of the feature instances. For this purpose, it is necessary to define topology packages that parallel the geometry packages in Clause 6. Figure 31 shows these packages and their dependencies.

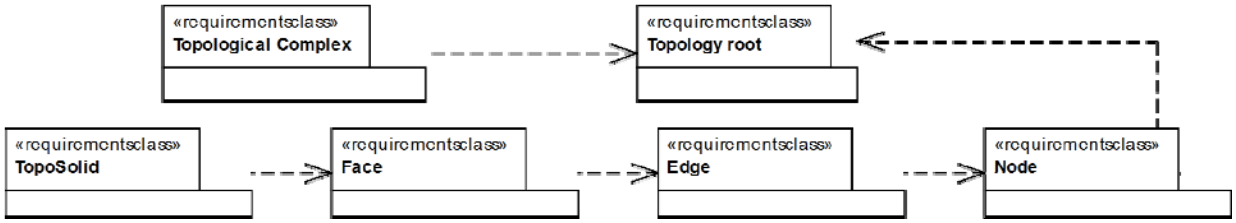



**Figure 31 — Topology packages and internal dependencies**

Figure 32 gives an overview of the class structure of the basic topological packages. The root class of the diagram is TopologyObject. Under this, there are Primitive, and Complex, which are related in way similar to the GeometryPrimitive and GeometryComplex, so that a Complex is an organized structure of Primitives. The major difference being that a GeometryPrimitive is more loosely coupled to a GeometryComplex, allowing it to stand alone, whereas a Primitive must be in at least one

© ISO 2013 – All rights reserved

Complex. An instance of DirectedTopo shall contain a reference to a Primitive and an orientation parameter, similar to the OrientablePrimitive in 6.3.13. Since only two orientations are possible, regardless of dimension, each primitive is associated to two directed topological entities similar to the relation between OrientableCurve and Curve, and between, OrientableSurface and Surface. To conserve on the number of objects and to make the natural identification of a primitive with its positive orientation, each primitive in each dimension is subclassed under its corresponding directed topological object.

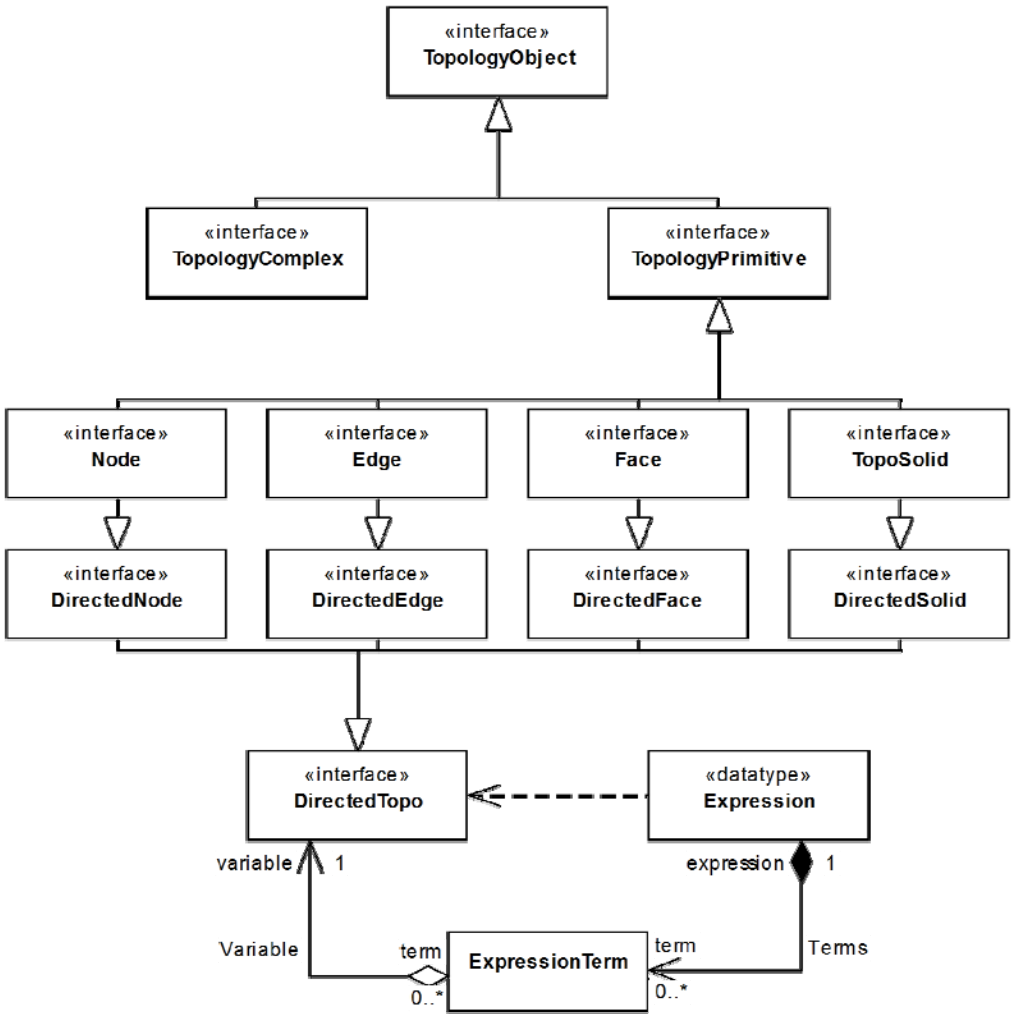



**Figure 32 — Topological class diagram**

## 11.2 Requirements class: Topology root

### 11.2.1 Semantics

Geometric calculations such as containment (point-in-polygon), adjacency, boundary, and network tracking are computationally intensive. For this reason, combinatorial structures known as topological complexes are constructed to convert computational geometry algorithms into combinatorial algorithms. Another purpose is, within the geographic information domain, to relate feature instances independently of their geometry. For the first purpose, topology definitions in this clause parallel the structure of the geometric definitions in Clause 6. For the second purpose, the classes in these packages are specified so that they can be used independently of the geometry.


© ISO 2013 – All rights reserved

A topological complex consists of collections of topological primitives of all kinds up to the dimension of the complex. Thus, a 2-dimensional complex must contain faces, edges, and nodes, while a 1-dimensional complex or graph contains only edges and nodes.

NOTE Topological primitives are equivalent to but are not subclasses of geometric primitives. This is consistent with the view that topological complexes are constructed to optimize computational geometry procedures by the use of combinatorial algorithms. This also permits the creation of structures that ignore geometric constraints by using a topological complex that is not realized by a geometric complex.

The key to understanding the use of computational topology is to see the related procedures in both systems. As Figure 33 shows, there is a great deal of parallelism between the ways in which primitives and complexes are related in the two class systems.

The topological system is based on algebraic manipulations of multivariate polynomials. The definitions of the procedures, functions, and operations in the topology packages are done so that geometric problems in the geometric domain can be translated into algebraic problems in the topology domain, solved there, and the solutions translated back to the geometric domain. A topological expression in this algebra is a multivariate, degree one polynomial, where the variables correspond to topological primitives.

The diagram in Figure 33 summarizes the relation between topology and geometry. The OCL constraint means that the diagram commutes such that navigation of the roles Primitive::complex followed by Complex.geometry is the same as navigation of the roles Primitive::geometry followed by GeometryPrimitive::complex.

NOTE A single GeometryPrimitive may be involved in many independent GeometryComplexes, each of which may be a realization of a different Complex. Thus, a GeometryPrimitive may be the realization of many different Primitives, since a Primitive must occur in one and only one maximal Complex (see 7.3.10.2). Since it is possible for an instantiable class to implement Primitive and Complex, or both GeometryPrimitive and Composite, it is possible that a particular instance of Primitive may be realized by a Composite, for example, see D.3.

© ISO 2013 – All rights reserved

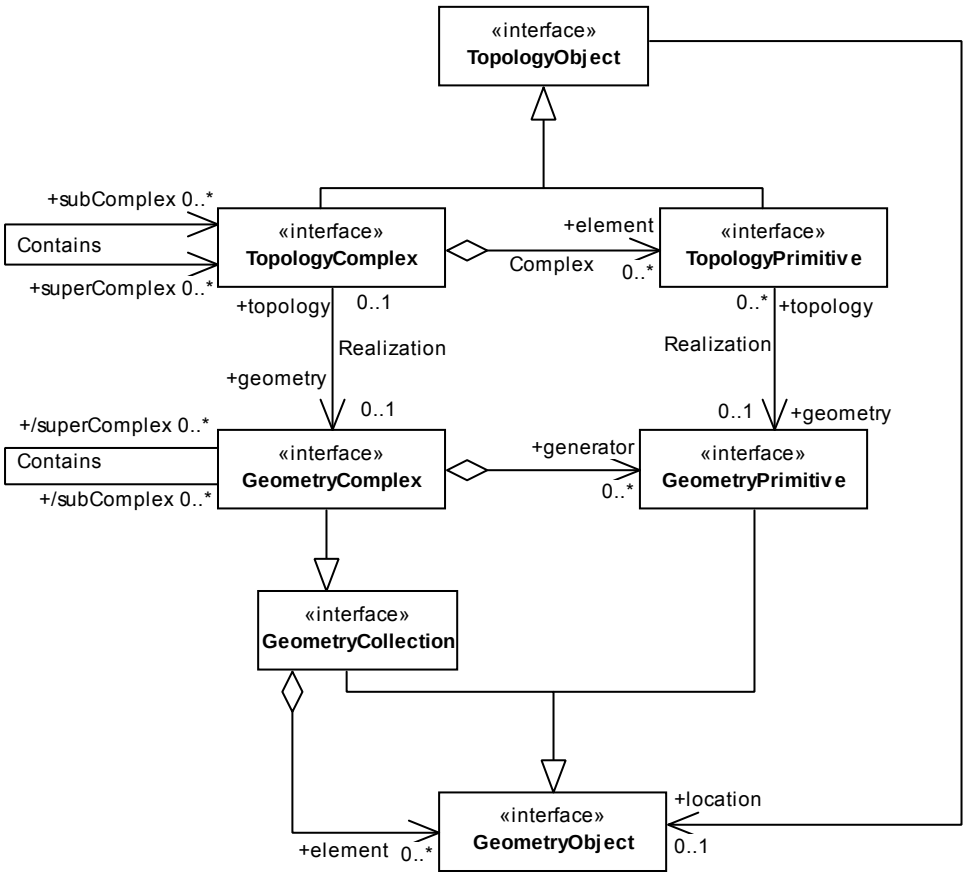



**Figure 33 — Relation between geometry and topology**

### 11.2.2 TopologyObject

#### 11.2.2.1 Semantics

Topological object, TopologyObject (Figure 34) is an abstract class that supplies a root type for topological complexes and topological primitives.

Logically and structurally, topological objects and geometric objects could share the same subclass structure, but since there is a categorical homomorphism from topology to geometry that preserves boundary operations, this approach could cause confusion between the boundary of a topological object and the boundary of the corresponding geometric object. While the two mechanisms share many computational characteristics, as demonstrated by the homomorphism, they are different operations and need to be clearly separated.


© ISO 2013 – All rights reserved

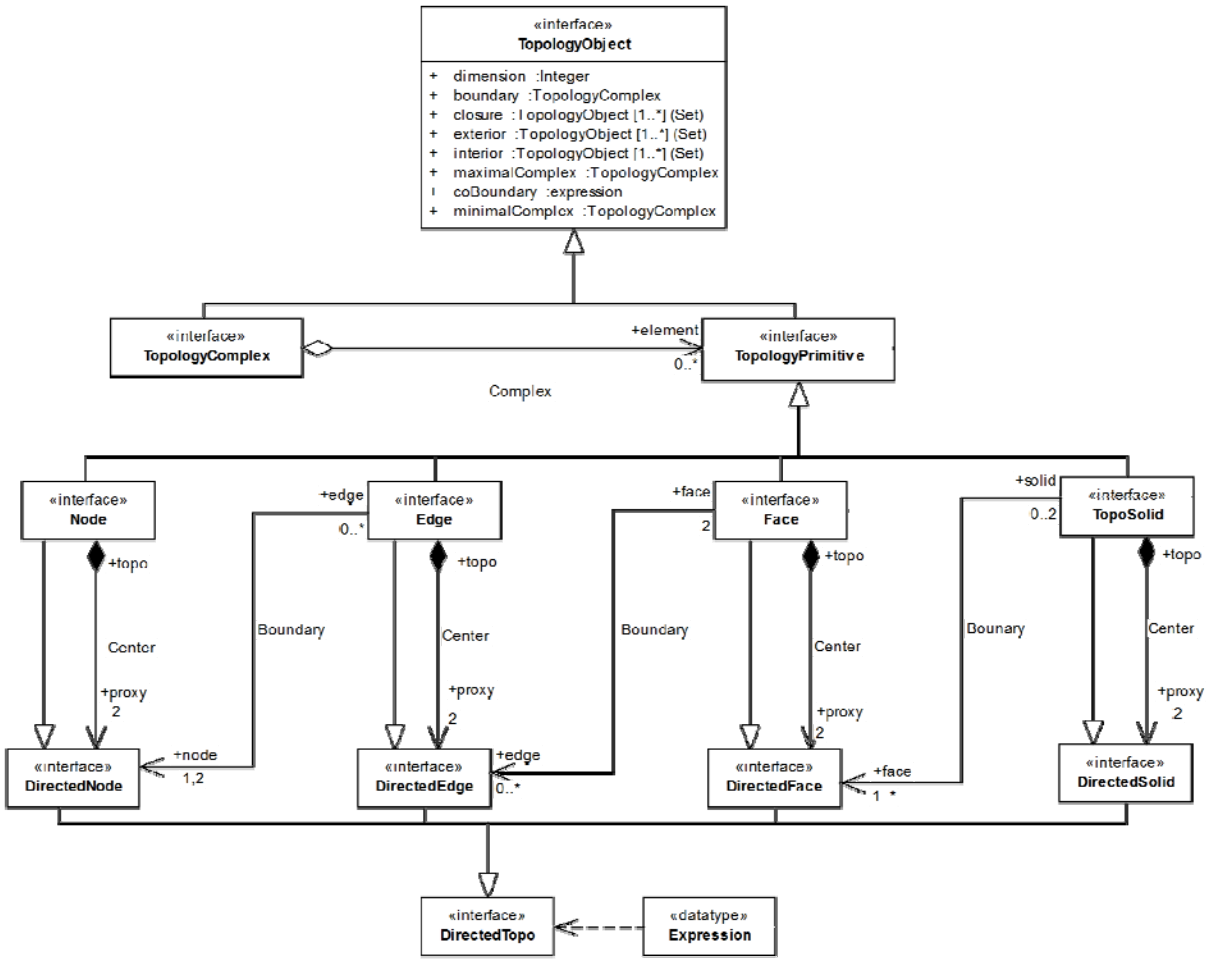



**Figure 34 — TopologyObject and classes**

### 11.2.2.2 Attribute TopologyObject: dimension

The integer returned by the operation "dimension" shall be the topological dimension of this TopologyObject. It shall be solely dependent on the instantiated class of the object and shall not be changed for a particular object without changing that object's class. For example, the value for dimension is 0 for nodes, one for edges, two for faces, and three for solids. Any GeometryObject associated to this TopologyObject shall have this same dimension.

```
TopologyObject::dimension():Integer
```

### 11.2.2.3 Attribute TopologyObject: boundary

The operation "boundary" shall return a set of DirectedTopo structured as a Boundary that represents the boundary of the TopologyObject.

```
TopologyObject::boundary():Boundary
```

If this TopologyObject is associated to a GeometryObject, its boundary shall be consistent in orientation with that GeometryObject as described in the geometry packages.

© ISO 2013 – All rights reserved

As a constraint, the dimension of a boundary shall always be one less than the dimension of the original object. For this reason, the dimension of the empty set shall be considered to be "–1".

```
TopologyObject:
boundary.dimension()=dimension()- 1
```

Figure 35 shows how the boundary function can be visualized as an association from objects of each dimension to objects of one less dimension.

In most cases the return value will be a valid value of an Expression. The boundary returned can fail to be a valid Expression because of the requirement for simplest terms. A dangling or isolated edge in a face (one that has the same face on both sides) would cancel out in the conversion to a topological expression.

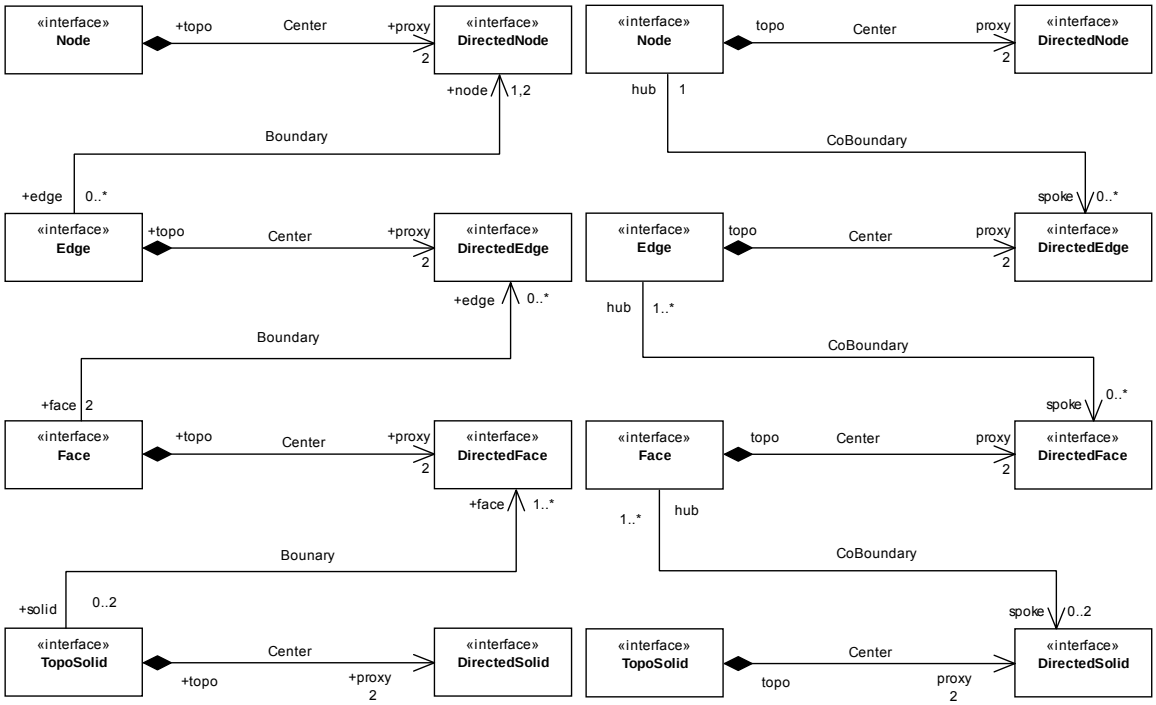



**Figure 35 — Boundary and coboundary operation represented as associations**

### 11.2.2.4 Attribute TopologyObject: coBoundary

The operation "coBoundary" shall return a Set of DirectedTopo that represents all the TopologyObjects that have this TopologyObject on their boundary.

In most cases the return value will be a valid value of a Expression (7.3.20). An exception to this is when the corresponding GeometryObject is on the boundary of a closed object (such as a curve that begins and ends at the same point). The TopologyObject corresponding to that GeometryObject would appear in the Set of DirectedTopo twice with opposite orientations and therefore cancel out when the coBoundary is cast from Set of DirectedTopo to Expression.

```
TopologyObject::coBoundary():Set<DirectedTopo>
```


© ISO 2013 – All rights reserved

Figure 36 illustrates how this operation can be visualized as a relation between dimension levels of the Primitives, similar to the boundary operation, but directed in the opposite direction, increasing dimension instead of reducing it.

#### 11.2.2.5 Attribute TopologyObject: interior

The operation "interior" shall return the finite set of Primitives that comprises the interior of this object within the maximal complex of this object. For a Primitive this will be a self-reference. For a Complex this will be all Primitive elements in the Complex not on the boundary of the Complex. This is the homomorphic equivalent of the interior of a geometric realization of this TopologyObject.

```
TopologyObject::interior():Set<Primitive>
```

#### 11.2.2.6 Attribute TopologyObject: exterior

The operation "exterior" shall return the finite set of Primitives that comprises the exterior of this object within the maximal complex of this object. This consists of all Primitives in the maximal Complex that are not in the interior or the boundary of this TopologyObject.

```
TopologyObject::exterior():Set<Primitive>
```

#### 11.2.2.7 Attribute TopologyObject: closure

The operation "closure" is often useful; it is defined as a union of the interior and boundary of an object, and is thus not required in a basic implementation.

```
TopologyObject::closure()=interior().union(boundary())
```

#### 11.2.2.8 Attribute TopologyObject: maximalComplex

The operation "maximalComplex ()" shall return the maximal Complex that contains this TopologyObject.

```
TopologyObject::maximalComplex():Complex
```

*A TopologyObject shall be included in one and only one maximal Complex.*

NOTE A complex is maximal if it is contained in no larger complex. The cardinality restriction implied by this operation means that any TopologyObject is in one and only one maximal complex.

### 11.2.3 Interface: TopologyPrimitive

#### 11.2.3.1 Semantics

Topological primitives, (Figure 36), are the non-decomposed elements of a topological complex. As such, they normally correspond to the geometric primitives of a like dimension that are the components of a geometric complex. When a geometric complex is the realization of a topological complex, then the primitives in each shall be in a dimension-preserving, 1-to-1 correspondence.

© ISO 2013 – All rights reserved

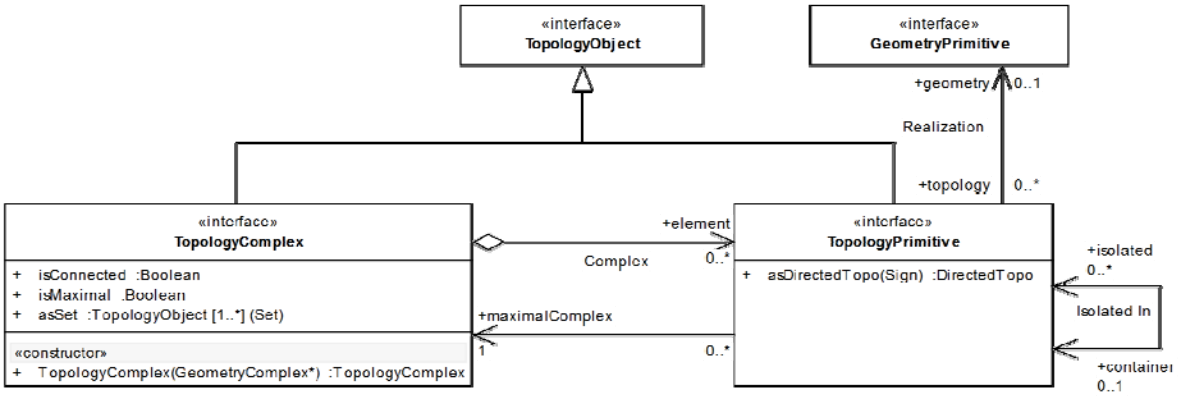



**Figure 36 — TopologyPrimitive**

### 11.2.3.2 Realization

The association "Realization" links this TopologicalPrimitive to the GeometryPrimitive that it represents in its maximal complex. If this TopologicalPrimitive is used to describe a logical topological structure that is not realized by a GeometryComplex, then this relationship shall be empty for all Primitives contained in this Primitive's maximal Complex. Each GeometryPrimitive may be associated to at most one Primitive in any Complex. If this Primitive is in any realized Complex, then it shall be associated to exactly one GeometryPrimitive. A GeometryPrimitive may be associated to different Primitives in different Complexes.

```
TopologicalPrimitive::geometry[0,1]:GeometryPrimitive
GeometryPrimitive::topology[0..*]:TopologicalPrimitive
```

NOTE Since Composites are subtyped under the corresponding primitives, it is possible to define a schema where the realization of a Primitive is a Composite of the same dimension. Thus a Edge can be realized as a CompositeCurve, see D.3.

### 11.2.3.3 Association: Complex

The association "Complex" shall link this Primitive to the finite set of Complexes that contain it. Every Primitive shall be in some number of Complexes that are all subcomplexes of a unique maximal Complex containing this Primitive.

```
TopologyPrimitive::complex[0…*]:Complex
TopologyComplex::element[0…*]:Primitive
```

### 11.2.3.4 Isolated In association

All of the adjacency relations in topology between primitives whose dimensions differ by one or 0 are handled by the boundary and coboundary operations. These operations only deal with instances of one primitive lying on the boundary of another primitive of one higher dimension, or with instances of the same dimension that share a common boundary element. This includes instances where a "dangling" edge has the same face on both sides, or a "dangling" face has the same solid on both sides. The exception to this is when one primitive is completely surrounded by a primitive of at least two higher dimensions, with no intermediate primitive. These are truly isolated. In faces, this includes nodes that are not attached to an intermediate edge on the boundary of that face. In a 3D space, the isolated node could be connected to another edge that is not on the boundary of the surface in question, such as in the case where the edge is realized by a curve perpendicular to the surface that


© ISO 2013 – All rights reserved

the face realizes. In solids, this can include nodes or edges that are not attached to surfaces in the boundary of the solid.

```
TopologyPrimitive::isolated[0..n]:Primitive
TopologyPrimitive::container[0,1]:Primitive
TopologyPrimitive:
   isolated.dimension()< self.dimension()- 1;
   container.count=0 implies
        Primitive→exists(boundary().topo→includes(self))
```

### 11.2.4 Interface: DirectedTopo

#### 11.2.4.1 Semantics

From a computational point of view, elements of DirectedTopo (Figure 32, Figure 34, Figure 38) are equivalent to the various orientable geometric objects (OrientableObject) in the geometry packages (OrientableCurve and OrientableSurface). DirectedNode and DirectedSolid do not have separate geometric object equivalents.

As in the geometry, each topological primitive inherits from its corresponding directed topological primitive, but it satisfies more constraints. This means that Node is equivalent to a positive DirectedNode, a Edge to a positive DirectedEdge, etc.

NOTE An alternative type hierarchy would have separated Primitive and DirectedTopo, which would have entailed three objects for each primitive: the primitive itself, its equivalent positive directed topological primitive, and its reversal (a negative directed) topological primitive. This alternative is a valid implementation of the abstract types in this model, but it does not emphasize the logical equivalence of a topological primitive and its positive directed topological primitive. From an algebraic point of view, the subclassing and OCL constraints that identify a primitive with its positive directed primitive make it equivalent to the standard interpretation of the unary "+" (plus) in algebra as in "x = + x". Since the most powerful use of topological objects is in their symbolic manipulation, maintaining an algebraic metaphor is appropriate.

There is an implicit relation between the directed topological objects of adjacent dimensions. The boundary and coboundary operations and relations use them to carry the same orientation sense. Thus if a positive directed edge is on the boundary of a face, then the positive directed face is on the coboundary of the associated edge. If a positive directed node is on the boundary of an edge, then the corresponding positive directed edge is on the coboundary of the associated node.

#### 11.2.4.2 Attribute: orientation

The attribute "orientation" shall be the sense in which this directed topological object is related to its underlying Primitive.

```
DirectedTopo::orientation:Sign="+"
```

#### 11.2.4.3 Operation: negate

The operation "negate" shall return the opposite orientation of this primitive.

```
DirectedTopo::negate():DirectedTopo
```

#### 11.2.4.4 Operation: asExpression

The operation "asExpression" shall create an Expression from this DirectedTopo, and shall retain the sign and the sense of the orientation. This operator shall be the constructor from the class Expression.

© ISO 2013 – All rights reserved

```
DirectedTopo::asExpression():Expression
```

#### 11.2.4.5 Association : Center

*The role "topo" in the association "Center" shall identify the associated Primitive.*

*The inverse role "proxy" shall identify the two DirectedTopo instances associated to the particular Primitive.*

```
DirectedTopo::topo[1]:TopolgyPrimitive
Primitive::proxy[2]:DirectedTopo
```

#### 11.2.4.6 Constraints

*Following the logic of the semantics of directed topological objects, the associated topology for each directed topological object shall be of the appropriate type.*

```
DirectedNode:
     topo.isKindOf(Node);
DirectedEdge:
     topo.isKindOf(Edge);
DirectedFace:
     topo.isKindOf(Face);
DirectedSolid:
     topo.isKindOf(Solid);
```

NOTE These constraints use the OCL operator "isKindOf" to indicate that the class of a directed topological primitive corresponding to a topological primitive must be a realization of the corresponding topological primitive type.

The Center association forms an important part of the algebra of the boundary and coBoundary operations.

```
DirectedTopo:
[boundary()=(orientation)*topo.boundary()]
Primitive:
[boundary()=(proxy.orientation)*proxy.boundary()]
DirectedTopo:
     negate.topo=topo;
     negate.orientation <> orientation;
```

### 11.2.5 Interface: Expression

#### 11.2.5.1 Semantics

Algebraic or computational topology is most easily conceptualized as the manipulation of multivariate, degree-one polynomials where the variables correspond to Primitives. The DirectedTopo class represents the terms in this algebra. The Expression class (Figure 37) represents the polynomial expressions.

The order of the terms in a polynomial does not affect its value, so the Expression class has been subclassed from Set<DirectedTopo>. The operations of the Expression class are those needed to construct, manipulate, and test these "polynomials".


© ISO 2013 – All rights reserved

The key to computational topology is the ability to treat pieces of topology in an algebraic or combinatorial manner. The primitives in this algebra are the Primitives. The monomials (single variable, single term polynomials) are the instances of Primitives, each with an integer coefficient, instantiated as ExpressionTerm.

Any constraint that would be consistent for multivariate, first-order polynomial algebra shall be valid for Expression, such as:

```
DirectedTopo:
     negate.expression()=expression().negate()
     asExpression.negate().plus(asExpression).isZero()
```

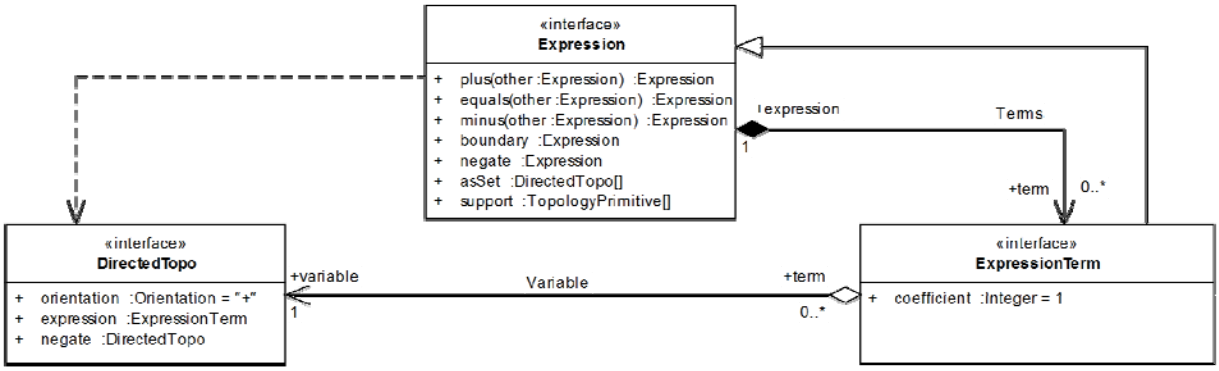



**Figure 37 — Expression**

### 11.2.5.2 Operation: plus(expression)

The operation "plus" acts as vector addition for Expressions.

*It shall combine DirectedTopo elements that have the same underlying instances of Primitive by adding their "orientation" coefficients. It shall remove any terms with zero-value coefficient.*

```
Expression::plus(other:Expression):Expression
```

### 11.2.5.3 Operation: minus

The operation "minus" acts as vector subtraction for Expressions. It shall combine DirectedTopo elements that have the same underlying instances of Primitive by subtracting their "orientation" coefficients. It shall remove any terms with zero coefficients.

```
Expression::minus(s:Expression):Expression
```

### 11.2.5.4 Operation: negate

*The operation "negate" shall negate each of the terms in the Expression. It is the unary minus operator for the polynomials.*

```
Expression::negate:Expression
```

© ISO 2013 – All rights reserved

#### 11.2.5.5 Operation: boundary

*The operation "boundary" shall replace each Primitive in each DirectedTopo in this Expression with its boundary represented as sequence of DirectedTopo objects and shall simplify the resultant expression. Boundaries always consist of Primitives of one lower dimension.*

*If the dimension of all the Primitives in this Expression is zero (the Primitives are all nodes), then the boundary operation shall return a zero Expression (no terms).*

```
Expression::boundary:Expression
```

#### 11.2.5.6 Operation: coBoundary

*The operation "coBoundary" shall replace each Primitive in each DirectedTopo in this Expression with its coBoundary and shall simplify the resultant expression.*

Coboundaries always consist of Primitives of one higher dimension. If the underlying geometry is not the boundary of anything, the coboundary is empty, which is zero as an expression.

*If the dimension of all the Primitives in this Expression is the same as the dimension of the corresponding maximal Complex, then the coBoundary operation shall return a zero Expression.*

```
Expression::coBoundary:Expression
```

#### 11.2.5.7 Operation: equals

The operation "equals" shall return TRUE for a polynomial equality. The order of the elements (terms) is not significant.

```
Expression::equals(other:Expression):Boolean
```

#### 11.2.5.8 Operation: support

The operation "support" shall cast this Expression as a set of Primitives for use in calculating geometric operators. The operation is essentially the "asSet" operation followed by a traversal of the Center association between DirectedTopo and Primitive. If the Expression has no terms after simplifications, then the support is the empty set "∅."

```
Expression::support:TopologyPrimitive[]
```

#### 11.2.5.9 Operation: asSet

The operation "asSet" shall cast this Expression as a set of DirectedTopo for use in calculating geometric operators. This cast shall include adding all boundary elements to the set until DirectedNodes are reached. In other words, the support of a Expression shall be a valid Complex. . If the Expression has no terms after simplifications, then the support is the empty set "∅."


© ISO 2013 – All rights reserved

```
Expression::asSet:DirectedTopo[]
```

### 11.2.6 ExpressionTerm

Expressions, like polynomials, consist of a set of terms, which consist of a variable and a coefficient. The expressions act like vectors with the DirectedTopo as basis vectors. The coefficients are usually integers in most applications.

```
ExpressionTerm=<coefficient:Integer = 1, variable: DirectedTopo>
```

Arithmetic shall be consistent with normal vector manipulation.

## 11.3 Requirements class: Node

### 11.3.1 Semantics

The Topological primitive package contains all the primitives for each dimension and supports classes for representations of their structural relationships.

© ISO 2013 – All rights reserved

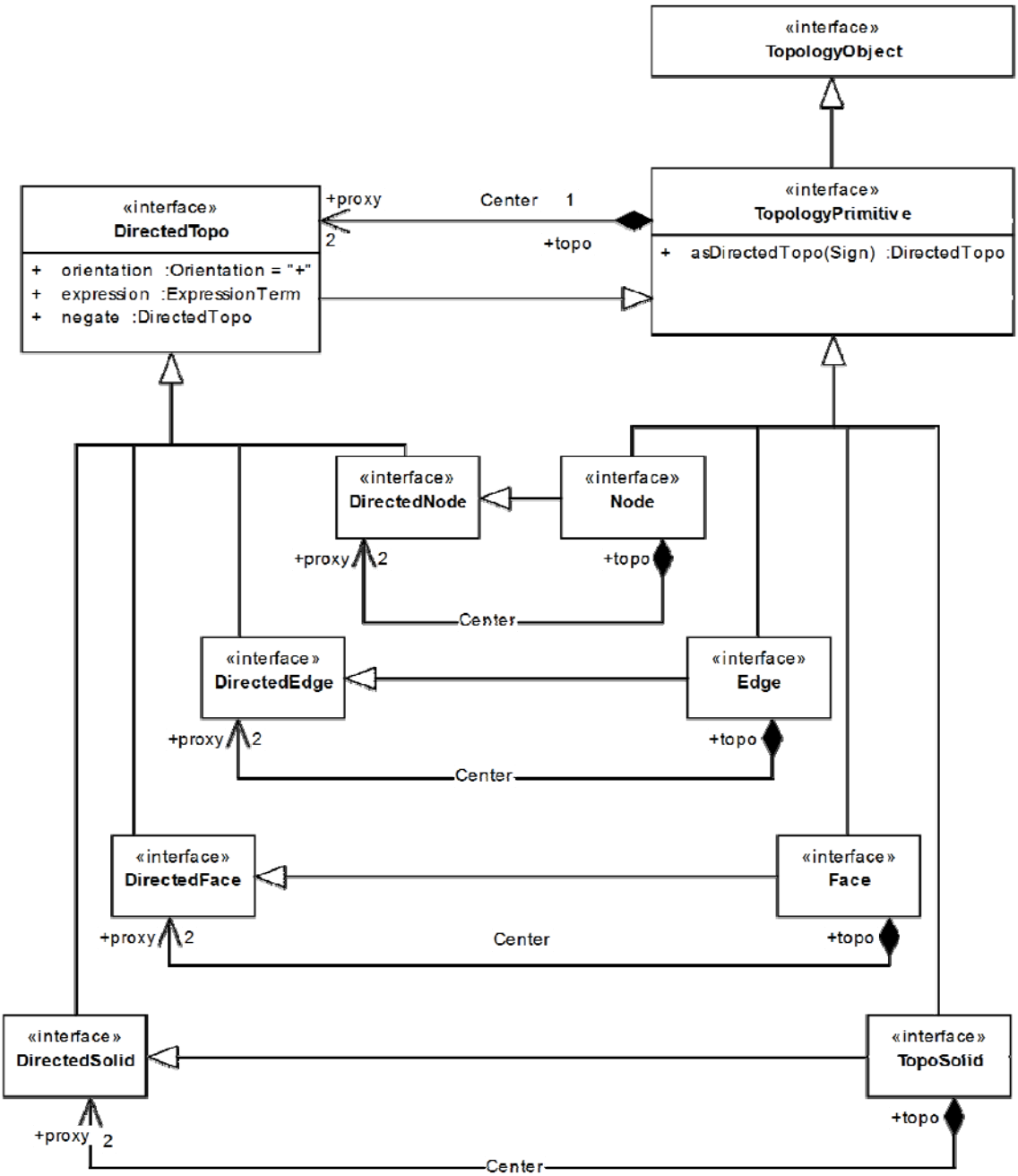



**Figure 38 — DirectedTopo subclasses**

### 11.3.2 Interface: Node

#### 11.3.2.1 Semantics

Node (Figure 41) inherits all of its interfaces from Primitive, with some elaboration on the structure of boundary and coboundary.

For Node, the operation “coBoundary” defined at TopologyObject shall always return a set of references to DirectedEdges indicating which edges enter (positive DirectedEdges) and which leave (negative DirectedEdges) the node. This operation is overridden from TopologyObject. The same information may be represented as an association.


© ISO 2013 – All rights reserved

NOTE In 2-dimensional maximal Complex containing this Node, the coBoundary may be sorted as a clockwise circular sequence in any geometric realization of this maximal Complex. In a 3D complex, the ordering is arbitrary.

```
Node::coBoundary:Set<DirectedEdge> {size=[0..n]}
Node::coBoundary.spoke:Set<DirectedEdge> {size=[0..n]}
```

#### 11.3.2.2 Center association

Each Primitive, including Node, is associated to two DirectedTopo instances.

```
Node::proxy[2]:DirectedNode
DirectedNode[1]:Reference<Node>
```

#### 11.3.2.3 boundary

The boundary operation for Node shall overrides that defined at TopologyObject by specifying the Empty set.

```
Primitive::boundary():NULL
```

#### 11.3.2.4 Constraints

The Node's dimension shall be 0, and its boundary is empty (NULL).

```
Node:
     TopologyObject::dimension=0;
     TopologyObject::boundary()=NULL;
```

NOTE A node may still be isolated in a face and be the end of an edge, as long as that edge is not on the boundary of the containing face. The geometric realization of this would be a curve that dangles in space, but terminates at its intersection with a surface.

### 11.3.3 Interface: DirectedNode

The class "DirectedNode" supports Node in the computational topology class Expression. For Node, the operation "boundary" defined at TopologyObject shall always return a zero-valued expression, corresponding to empty geometry. This operation is overridden from TopologyObject.

```
Node::boundary():NULL
```

## 11.4 Requirements class: Edge

### 11.4.1 Interface: Edge

#### 11.4.1.1 Semantics

The primitive Edge (Figure 42) is the 1-dimensional primitive for topology. For Edge, the operation "boundary" defined at TopologyObject shall return a pair of nodes, one at the start of the edge (negative DirectedNode) and one at the end (positive DirectedNode). This operation is overridden from TopologyObject. The same information may be represented as an association.

```
Edge::boundary():Set<DirectedNode> {size=2}
Edge::boundary.boundary:Set<DirectedNode> {size=2}
```

© ISO 2013 – All rights reserved

#### 11.4.1.2 coBoundary

For Edge, the operation "coBoundary" defined at TopologyObject shall return a circular sequence of directed faces indicating which faces use this edge (positive DirectedFace) or its negative proxy (negative DirectedFace) on their boundary. The circular sequence shall represent a clockwise enumeration of these faces as viewed from the end point of the associated curve in any geometric realization of the maximal Complex in which this Edge is contained. This operation is overridden from TopologyObject. The same information may be implemented as an association.

```
Edge::coBoundary():CircularSequence<DirectedFace> {size=[0..n]}
Edge::coBoundary.spoke:CircularSequence<DirectedFace>
     {size=[0..n]}
```

NOTE In the 2-dimensional planar case, the coboundary has at most two faces. In the full topology case, there are precisely 2, one directed face having a positive "+" orientation and the associated face lying to the left of the edge, and the other directed face having a negative "-" orientation, and the associated face lying to the right of the edge.

#### 11.4.1.3 boundary

The boundary operation for Edge shall overrides that defined at TopologyObject by specifying a EdgeBoundary, consisting of a start node and end node.

```
Edge::boundary():EdgeBoundary
```

The Edge shall also has an association Boundary with association role boundary which specifies this same information as two directed edges, oriented positively for the end node and negatively for the start node.

```
Edge::boundary[2]:DirectedNode
```

#### 11.4.1.4 Center association

Each Primitive, including Edge is associated to two DirectedTopo instances.

```
Edge::proxy[2]:DirectedEdge
DirectedEdge::topo[1]:Reference<Edge>
```

NOTE In the 2-dimensional planar case, each directed edge bounds at most one face, precisely one face in a full planar topology. In the 3-dimensional case, or in a non-planar 2D complex, a directed edge can bound several faces.

#### 11.4.1.5 Constraints

The Edge shall have dimension 1.

```
Edge:
     TopologyObject::dimension()=1
```

### 11.4.2 Interface: DirectedEdge

The class "DirectedEdge" supports Edge in the computational topology class Expression. It is analogous to the concept of a OrientableCurve, in the sense that it acts as a proxy for the base curve/edge when needed.


© ISO 2013 – All rights reserved

## 11.5 Requirements class: Face

### 11.5.1 Interface: Face

#### 11.5.1.1 Semantics

The class “Face” (Figure 43) provides topological primitives for Surface.

#### 11.5.1.2 boundary

For Face, the operation “boundary” defined at TopologyObject shall return a set of directed edges with appropriate orientation. This operation is overridden from TopologyObject. The same information may be represented as an association.

```
Face::boundary():FaceBoundary
```

NOTE The same restriction on the meaning of exterior applies to the topology as did to the geometry.

The Face shall also has an association Boundary with association role boundary that specifies this same information as directed edges, oriented positively for the left side of the edge and negatively for the right.

```
Face::boundary[1..*]:DirectedEdge
```

The additional information that is returned by the boundary operator is the organization of the FaceBoundary into rings and an indication as to which ring is the exterior.

#### 11.5.1.3 coBoundary

For Face, the operation “coBoundary” defined at TopologyObject shall return a set of references to directed solids indicating which solids use this face (positive DirectedSolid) or its negative proxy (negative DirectedSolid) on their boundary. This operation is overridden from TopologyObject. The same information may be implemented as an association.

```
Face::coBoundary()[0..2]:Reference<DirectedSolid>
Face::coBoundary.spoke[0..2]:Reference<DirectedSolid>
```

#### 11.5.1.4 Center association

Each Primitive, including Face is associated to two DirectedTopo instances.

```
Face::proxy[2]:DirectedFace
DirectedFace::topo[1]:Reference<Face>
```

#### 11.5.1.5 Constraints

Face’s dimension shall be 2.

```
Face:
     Face:TopologyObject::dimension=2
```

© ISO 2013 – All rights reserved

### 11.5.2 Interface: DirectedFace

DirectedFaces shall be used in defining the boundary of a Solid. It is analogous to the concept of a OrientableSurface, in the sense that it acts as a proxy for the base surface/face when needed.

## 11.6 Requirements class: TopoSolid

### 11.6.1 Interface: TopoSolid

#### 11.6.1.1 Semantics

The class "TopoSolid" (Figure 44) provides topological primitives for Solid.

#### 11.6.1.2 boundary

For Solid, the operation "boundary" defined at TopologyObject shall return a collection of faces or their negative proxies. This operation is overridden from TopologyObject. The same information may be represented as an association.

```
TopoSolid::boundary():SolidBoundary
```

The Solid shall also has an association Boundary with association role boundary that specifies this same information as directed edges, oriented positively for below the face and negatively for above the face.

```
TopoSolid::boundary[1..*]:DirectedFace
```

The additional information that is returned by the boundary operator is the organization of the SolidBoundary into shells and an indication as to which shell is the exterior.

#### 11.6.1.3 coBoundary

For Solid, the operation "coBoundary" shall return NULL.

```
TopoSolid::coBoundary():NULL
```

#### 11.6.1.4 Center association

Each Primitive, including Solid is associated to two DirectedTopo instances.

```
TopoSolid::proxy[2]: DirectedSolid
DirectedSolid::topo[1: TopoSolid
```

### 11.6.2 Interface: DirectedSolid

The class "DirectedSolid" supports Solid in the computational topology class Expression.

## 11.7 Requirements class: Topological complex

### 11.7.1 Semantics

The package "Topological complex" provides additional classes for the creation of Complexes.


© ISO 2013 – All rights reserved

### 11.7.2 Interface: Complex

#### 11.7.2.1 Semantics

This clause contains the definition of topological complexes that parallel the geometric complexes introduced earlier in 6.6. A Complex (Figure 46) may use set operations on its elements to perform the equivalent set operations on the underlying sets of DirectPositions that are represented by the geometric elements of a geometric realization (a GeometryComplex).

#### 11.7.2.2 Complex: constructor of a topological complex

The default construction of a topological complex shall be to generate it from a geometric complex. After the construction, the geometric complex shall be the geometric realization of the topological complex. Only geometric complexes that consist of mutually disjoint geometric primitives will generate a topological complex without error.

```
Complex::Complex(GC:GeometryComplex):Complex
```

The use of the default constructor to define a default topological complex for each geometric complex assures that the topology represented by the Complex is the topology of a geometric configuration as represented by the GeometryComplex. The association "Realization" shall trace each part of the Complex back to the appropriate part of the GeometryComplex. This allows us to speak of topological operations within a topological complex (Complex) as if they occurred directly on a geometric complex (GeometryComplex).

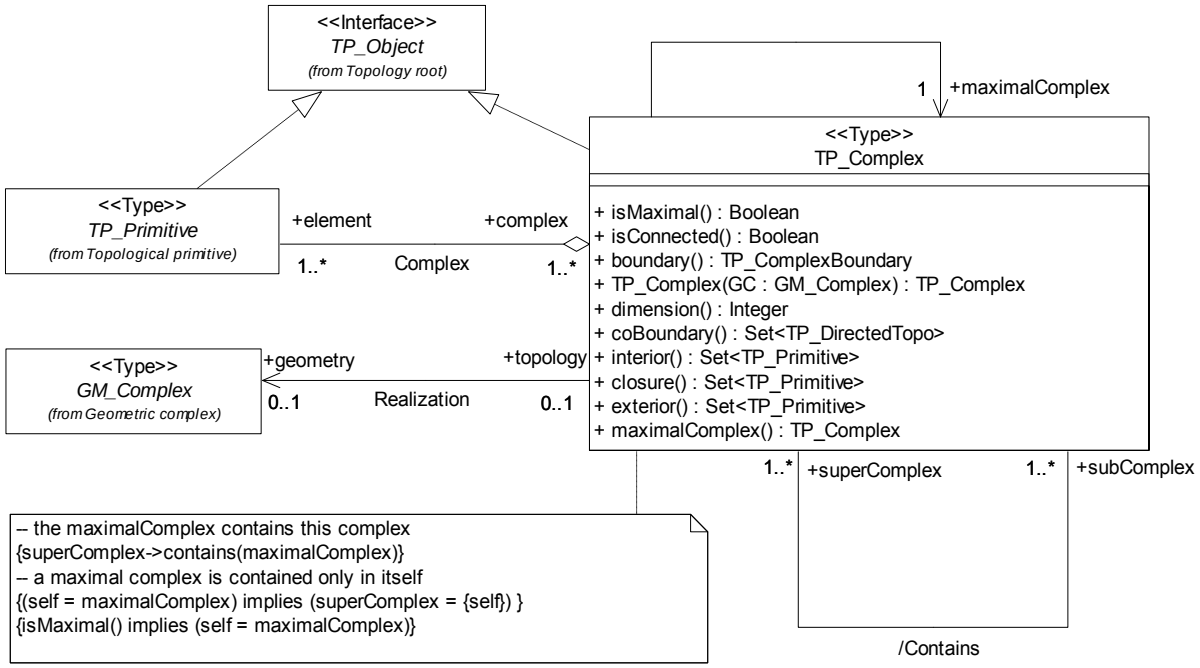



**Figure 39 — Complex**

© ISO 2013 – All rights reserved

#### 11.7.2.3 maximalComplex

The private attribute “maximalComplex” contains a reference to the unique maximal topological complex of which this Complex is a member. This is needed for encoding to determine the limits of an export data set.

- `Complex::maximalComplex:Reference<Complex>`

#### 11.7.2.4 isMaximal

The Boolean operation “isMaximal” shall return TRUE if this Complex is contained in no larger Complex.

```
Complex::isMaximal():Boolean
```

#### 11.7.2.5 isConnected

The Boolean valued operation “isConnected” shall return TRUE if this Complex is topologically connected.

```
Complex::isConnected():Boolean
```

NOTE If a Complex is connected, then its geometric realization is also connected. This does not imply that it is a composite (geometric or topological), since composites must comply with the stronger constraint of being isomorphic to a primitive. To test whether or not a topological complex is connected without referring to a geometric realization requires that the transitive closure of the boundary, coBoundary, and IsolatedIn associations be calculated. If every primitive in the complex is linked to every other primitive in the complex by a sequence of these association roles where each intermediate primitive is in the complex, then the complex is connected.

#### 11.7.2.6 Contains association

The derived association “Contains” shall describe which other Complexes are contained in this Complex as sets of Primitives. The “superComplex” role is the larger of the two complexes and the “subComplex” role is the smaller. This relation shall be consistent with the “contains” operation inherited from Set<Primitive>.

```
Complex::subComplex[1..n]:Reference<Complex>
Complex::superComplex[1..n]:Reference<Complex>
```

#### 11.7.2.7 Complex association

The “Complex” association shall relate the Primitive elements to this Complex. It is this association that makes the Complex a Set<Primitives>. The set operations implied by “Contains” should be consistent with this definition of the Complex as a set of primitives.

```
Complex::element[1..n]:Reference<Primitive>
Primitive::complex[[1..n]:Reference<Complex>
```

#### 11.7.2.8 Realization association

The realization association links this Complex to its corresponding GeometryComplex (if any).


© ISO 2013 – All rights reserved

```
Complex::geometry[0,1]:GeometryComplex
GeometryComplex::topology[0,1]:Complex
```

# 12 Derived topological relations

## 12.1 Introduction

This is a specification for characterizing topological relations as operators for query. These query operators can be calculated using the set theoretic operations defined on GeometryObject and its subtypes and on algebraic operations defined on Expression. These two mechanisms are equivalent for geometric complexes that are realizations of the corresponding topological complexes. The operators defined in this clause are meant mainly for query evaluation and are defined in such a manner as to allow a variety of implementations to be assured of equivalent results against datasets with equivalent information content.

This International Standard assigns specific names to particular spatial operators that have been commonly used in standards. Application may use any of these common operators and may define application specific operations defined using the formal techniques. Application schemas may use any or all of the following three classification techniques to specify application specific operators. In the cases below, the classification scheme is based on the class TopologyObject. This also defines the same operators on the class GeometryObjects given that the restrictions defined above for the creation of Complexes from collections of GeometryObjects are followed. What is to follow is valid for point, curve, surface, and solid objects. The theories for aggregate objects fall into two general categories based on the definition of the boundary of aggregate objects. There is yet to be a general consensus on which theory is more appropriate, and as long as an application specifies which it is using, this will not be an issue. If the proper information is transmitted with these operation definitions, including the type of aggregate boundary used, and the classification, the repeatability of these operators can be guaranteed on compatible topological data sets.

The conformance of a query system to this part of this International Standard shall mean that the supported topological query operations can be defined according to the characterizations laid out in one of the subsequent clauses and that all operators defined in the clause can be made available directly or through a well understood combination of supported operators. Minimal compliance to this clause implies that:

1) Boolean query operations are defined in terms consistent with the included subclauses.
2) All valid Boolean operators definable within the context of one or more of the 8.2, 8.3, or 8.4 are available for use.
3) If aggregate collections are used, the choice of boundary operation is specified.

Complete compliance requires support of all of the valid Boolean operators definable within the context of this entire clause.

## 12.2 Requirements class: Boundary operators for aggregate objects

### 12.2.1 Common semantics

In both boundary definitions below, the union of all of the geometric objects is first represented by a Geometry Complex (Clause 6.3.19) in such a manner as each object is a subcomplex of the created complex. This complex is then converted to an equivalent topological complex. Each geometric object is thus represented as a collection of topological objects.

© ISO 2013 – All rights reserved

### 12.2.2 "Mod 2" boundaries

Mod 2 boundaries are associated to Expressions that use mod 2 integer coefficients instead of integers. Mod 2 integer arithmetic uses two symbols {0, 1} and a "+" operator defined by:

0+0=0; 1+0=1; 1+1=0; -0=0; -1=1

If an Expression for the boundary of a geometry object is calculated using integer-based expressions as defined in Clauses 11.2.5 and 11.2.6, then any even coefficient is set to zero and any odd coefficient is set to 1. The support of this expression is the boundary.

Another way to state this is to use the complex defined above. The interior of a collection of elements includes the interior of all elements. Any point on an odd number of boundaries of the geometries in the complex is in the boundary of the complex. Any point on an even number of boundaries of elements in the complex is in the interior of the complex.

### 12.2.3 At least 2 boundaries

The same process as in the Mod 2 boundary, except any point on only 1 boundary of an element is on the boundary of the complex, and any point on 2 or more boundaries is interior to the compex.

## 12.3 Requirements class: Boolean or set operators

### 12.3.1 Form of the Boolean operators (originally Egenhofer 4 matrix)

Set theoretic operators are sometimes referred to as Boolean operators. Since such operators do not distinguish between the interior and boundary of a set, the closure operation is used to combine them:

```
GeometryObject::closure():==interior().union(boundary())
```

For two objects, A and B the following four intersection operations may be done (c=closure of, e=exterior of):

$$\begin{bmatrix} cA \cap cB & cA \cap eB \\ eA \cap cB & eA \cap eB \end{bmatrix}$$

This matrix of sets may be tested to see if each set is empty or not. This classifies the relationship between A and B into one of $2^4$, or 16, classes.

$$\begin{bmatrix} \mathrm{c}(A^c \cap B^c) & c(A^c \cap B^e) \\ \mathrm{c}(A^e \cap B^c) & c(A^e \cap B^e) \end{bmatrix} \text{ where } c(X) = \begin{cases} False & \text{if } X = \varnothing \\ True & \text{if } X \neq \varnothing \end{cases}$$

An operator may be defined as a template that is applied to the intersection matrix to test for a particular spatial relationship between the two objects. The template is a matrix of four extended Boolean Values whose interpretation is given in Table 8. There are $3^4$ or 81 possible operator templates.


© ISO 2013 – All rights reserved

**Table 2 — Meaning of Boolean intersection pattern matrix**

| Symbol | Non Empty? | Meaning |
|---|---|---|
| T | TRUE | The intersection at this position of the matrix is non-empty. |
| F | FALSE | The intersection at this position of the matrix is empty. |
| N | NULL | This operator does not test the intersection at this position of the matrix. |
| NOTE The value TRUE means the set is non-empty (see column header). | | |

To test if two objects are related in agreement with a particular operator template, the intersections not associated to NULL are calculated and tested for non-empty according to the pattern in the matrix. If there is agreement, the value of the operator for this pair of geometry objects is TRUE, and if not, the value is FALSE.

### 12.3.2 Boolean Relate

The operator "bRelate" shall return TRUE if these objects are spatially related by testing for intersections between the closure and exterior of the two geometric objects as controlled by the values in the patternMatrix.

```
Boolean bRelate(A:GeometryObject,B:GeometryObject,patternMatrix)
Boolean bRelate(A:TopologyObject,B:TopologyObject,patternMatrix)
```

The "patternMatrix" is listed as a string of 4 characters from T, F, or N, given in row major form, i.e., the two values for the first row, followed by the two for the second row of the matrix.

### 12.3.3 Relation to set operations

The Boolean relate can be used to implement the "contains", "intersects" and "equals" operations of GeometryObject defined in 6.2.2.18.

EXAMPLE

```
C:Composite, G:GeometryObject;
C.contains(G)=bRelate(C, G, "TNFT" );
```

## 12.4 Requirements class: Egenhofer operators

### 12.4.1 Form of the Egenhofer operators (originally Egenhofer 9 matrix)

For two objects, A and B the following 9 intersection operations may be done (see references [8] and [9]).

| | | |
|---|---|---|
| intersection[boundary(A),boundary(B)] | intersection[boundary(A),interior(B)] | intersection[boundary(A),exterior(B)] |
| intersection[interior(A),boundary(B)] | intersection[interior(A),interior(B)] | intersection[interior(A),exterior(B)] |
| intersection[exterior(A),boundary(B)] | intersection[exterior(A),interior(B)] | intersection[exterior(A),exterior(B)] |

More symbolically(c=closure of, e=exterior of, ∂=boundary of):

$$\begin{vmatrix} \partial A \cap \partial B & \partial A \cap iB & \partial A \cap eB \\ iA \cap \partial B & iA \cap iB & iA \cap eB \\ eA \cap \partial B & eA \cap iB & eA \cap eB \end{vmatrix}$$

© ISO 2013 – All rights reserved

This matrix of sets (called the 9 matrix) may be tested to see if each is empty or not. This classifies the relationship between A and B into one of $2^9$, or 512, classes. Actually, not all 512 are geometrically possible, but that is not of consequence to what is to follow.

An operator may be defined as a template that is applied to the intersection matrix to test for a particular spatial relationship between the two objects. The template is a matrix of nine extended Boolean Values whose interpretation is given in Table 9, the content of which is identical to the previous table. There are $3^9$ or 19 683 possible operator templates.

**Table 3 — Meaning of Egenhofer intersection pattern matrix**

| Symbol | Non Empty? | Meaning |
|---|---|---|
| T | TRUE | The intersection at this position of the matrix is non-empty. |
| F | FALSE | The intersection at this position of the matrix is empty. |
| N | NULL | This operator does not test the intersection at this position of the matrix. |

To test if two objects are related in agreement a particular operator; the intersections not associated to NULL are calculated and tested for non-empty according to the pattern in the matrix. If there is agreement, the value of the operator for these two objects is TRUE, and if not, the value is FALSE.

### 12.4.2 Egenhofer relate

The operator "eRelate" shall return TRUE if these objects are spatially related by testing for intersections between the interior, boundary and exterior of the two geometric objects as controlled by the values in the patternMatrix.

```
Boolean eRelate(GeometryObject, GeometryObject, patternMatrix)
Boolean eRelate(TopologyObject, TopologyObject, patternMatrix)
```

The "intersectionPatternMatrix" is listed as a string of nine characters (each being a T, F, or N) in row major form.

### 12.4.3 Relation to set operations

The Egenhofer relate can be used to implement the "contains", "intersects" and "equals" operations of GeometryObject defined in 6.2.2.18.

EXAMPLE

```
C:GeometryPrimitive, G:GeometryPrimitive;
C.contains(G)=eRelate(C, G, "NFNNTNNFT" );
C:GeometryPrimitive, G:Composite;
C.contains(G)=eRelate(C, G, "FFNTTNFFT" );
```

## 12.5 Requirements class: Full topological operators

### 12.5.1 Form of the full topological operators

The full topological operators take dimension differences into account (see references [4] and [5] for further analysis of this extension) and are done in a manner similar to the Egenhofer operators, but a finer distinction is made on the possible values.


© ISO 2013 – All rights reserved

**Table 4 — Meaning of full topological intersection pattern matrix**

| Symbol | Non Empty? | Meaning |
|---|---|---|
| 0 | TRUE | The intersection at this position of the matrix contains only points. |
| 1 | TRUE | The intersection at this position of the matrix contains only points, and curves. |
| 2 | TRUE | The intersection at this position of the matrix contains only points, curves, and surfaces. |
| 3 | TRUE | The intersection at this position of the matrix contains only points, curves, surfaces and solids. |
| F | FALSE | The intersection at this position of the matrix is empty. |
| N | NULL | This operator does not test the intersection at this position of the matrix. |

To test if two objects are related in agreement with one of the possible $6^9 = 10\ 077\ 696$ operator templates, the intersections not associated to NULL are calculated and tested for non-empty and dimension, according to the pattern in the matrix. If there is agreement, the value of the operator for these two objects is TRUE, and if not, the value is FALSE.

### 12.5.2 Full topological relate

The operator "cRelate" shall return TRUE if these objects are spatially related by testing for intersections between the interior, boundary and exterior of the two geometric objects as controlled by the values in the patternMatrix.

```
Boolean cRelate(GeometryObject, GeometryObject, patternMatrix)
Boolean cRelate(TopologyObject, TopologyObject, patternMatrix)
```

The "patternMatrix" is listed as nine characters (each being a 0, 1, 2, 3, F, or N) in row major form.

## 12.6 Requirements class: Combinations

Operators may be defined as any Boolean combination of one or more of the primitive operations in the preceding sections.

ED: Additional clauses will be added to this section to describe sets of named Boolean operators in use, from Simple Features and others such as RC8 and RC16.
Either an informative clause or new requirements class may be added here to create sets of name operators whose definitions are based on the above mechanisms (and dependent on the definition of boundary used).

© ISO 2013 – All rights reserved

# Annex A (normative)Abstract test suite

Ed: Needs updated to follow new set of requirements classes.

## A.1 Conformance class: Coordinate:Test

1. Test Purpose: test all requirements in the requirement class Coordinate (Clause 6.2).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: (Clause 6.2).
4. Test Type: Capability.

## A.2 Conformance class: Geometry

### A.2.1 Dependency Coordinate

1. Test Purpose: test Conformance class: Coordinate (Clause A.1)
2. Test Method: Use method of referenced class.
3. Reference: Clause A.1
4. Test Type: Capability.

### A.2.2 Test Geometry

1. Test Purpose: test all requirements in the requirement class Geometry (Clause 6.3).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 6.3.
4. Test Type: Capability.

## A.3 Conformance class: Lines

### A.3.1 Dependency Geometry

1. Test Purpose: test Conformance class: Geometry (Clause A.2)
2. Test Method: Use method of referenced class.
3. Reference: (Clause A.2)
4. Test Type: Capability.

### A.3.2 Test Lines

1. Test Purpose: test all requirements in the requirement class Lines (Clause 7.1).
2. Test Method: Inspect the documentation, the application schema or profile.
3. Reference: Clause 7.1.
4. Test Type: Capability.

## A.4 Conformance class: Geodesics

### A.4.1 Dependency Lines

1. Test Purpose: test Conformance class: Lines (Clause A.3)
2. Test Method: Use method of referenced class.
3. Reference: (Clause A.3)
4. Test Type: Capability.


© ISO 2013 – All rights reserved

### A.4.2 Test Geodesics

1. Test Purpose: test all requirements in the requirement class Geodesics (Clause7.2).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference Clause 7.4.
4. Test Type: Capability.

## A.5 Conformance class: Polynomials

### A.5.1 Dependency Geodesics

5. Test Purpose: test Conformance class: Geodesics (Clause A.4)
6. Test Method: Use method of referenced class.
7. Reference: (Clause A.4)
8. Test Type: Capability.

### A.5.2 Test Polynomials

5. Test Purpose: test all requirements in the requirement class Polynomials (Clause 7.3).
6. Test Method: Inspect the documentation of the application schema or profile.
7. Reference Clause 7.3.
8. Test Type: Capability.

## A.6 Conformance class: Conics

### A.6.1 Dependency Polynomials

1. Test Purpose: test Conformance class: Polynomials (Clause A.5)
2. Test Method: Use method of referenced class.
3. Reference: ClauseA.5.
4. Test Type: Capability.

### A.6.2 Test Conics

1. Test Purpose: test all requirements in the requirement class Conics (Clause 7.4).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 7.4.
4. Test Type: Capability.

## A.7 Conformance class: Spiral Curves

### A.7.1 Dependency Conics

1. Test Purpose: test Conformance class: Conics (Clause A.6).
2. Test Method: Use method of referenced class.
3. Reference: Clause A.6.
4. Test Type: Capability.

### A.7.2 Test Spirals

1. Test Purpose: test all requirements in the requirement class Conics (Spirals 7.5).
2. Test Method: Inspect the documentation of the application schema or profile.

© ISO 2013 – All rights reserved

3. Reference: Clause 7.5
4. Test Type: Capability.

## A.8 Conformance class: Spline Curve

### A.8.1 Dependency Polynomials

1. Test Purpose: test Conformance class: Polynomials (Clause A.5)
2. Test Method: Use method of referenced class.
3. Reference: ClauseA.5.
4. Test Type: Capability.

### A.8.2 Test Spline Curve

1. Test Purpose: test all requirements in the requirement class Spline Curve (Clause 7.6).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 7.6
4. Test Type: Capability.

## A.9 Conformance class: Gridded surfaces

### A.9.1 Dependency Geometry

1. Test Purpose: test Conformance class: Geometry (Clause A.2)
2. Test Method: Use method of referenced class.
3. Reference: (Clause A.2)
4. Test Type: Capability.

### A.9.2 Test Gridded surfaces

1. Test Purpose: test all requirements in the requirement class Gridded surfaces (Clause 8.1).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 8.1
4. Test Type: Capability.

## A.10 Conformance class: Polygon

### A.10.1 Dependency Geometry

1. Test Purpose: test Conformance class: Geometry (Clause A.2)
2. Test Method: Use method of referenced class.
3. Reference: (Clause A.2)
4. Test Type: Capability.

### A.10.2 Test Polygon

1. Test Purpose: test all requirements in the requirement class Polygon (Clause 8.2).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 8.1
4. Test Type: Capability.


© ISO 2013 – All rights reserved

## A.11 Conformance class: Conic Surface

### A.11.1 Dependency Gridded surfaces

1. Test Purpose: test Conformance class: Gridded surfaces (Clause A.9)
2. Test Method: Use method of referenced class.
3. Reference: Clause A.9
4. Test Type: Capability.

### A.11.2 Test Conic Surface

1. Test Purpose: test all requirements in the requirement class Conic Surface (Clause8.3).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 8.3
4. Test Type: Capability.

## A.12 Conformance class: Spline Surface

### A.12.1 Dependency Gridded surfaces

5. Test Purpose: test Conformance class: Gridded surfaces (Clause A.9)
6. Test Method: Use method of referenced class.
7. Reference: Clause A.9
8. Test Type: Capability.

### A.12.2 Test Spline Surface

1. Test Purpose: test all requirements in the requirement class Spline Surface (Clause 8.4).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 8.1
4. Test Type: Capability.

## A.13 Conformance class: Boundary Representation

### A.13.1 Dependency Geometry

1. Test Purpose: test Conformance class: Geometry (Clause A.2)
2. Test Method: Use method of referenced class.
3. Reference: (Clause A.2)
4. Test Type: Capability.

### A.13.2 Test Boundary Representation

1. Test Purpose: test all requirements in the requirement class Boundary Representation (Clause 9.1).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 9.1
4. Test Type: Capability.

© ISO 2013 – All rights reserved

## A.14 Conformance class: Gridded Solid

### A.14.1 Dependency Geometry

1. Test Purpose: test Conformance class: Geometry (Clause A.2)
2. Test Method: Use method of referenced class.
3. Reference: (Clause A.2)
4. Test Type: Capability.

### A.14.2 Test Spline Gridded Solid

1. Test Purpose: test all requirements in the requirement class Gridded Solid (Clause 9.2).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 9.2
4. Test Type: Capability.

## A.15 Conformance class: Geometric complex

### A.15.1 Dependency Geometry

1. Test Purpose: test Conformance class: Geometry (Clause A.2)
2. Test Method: Use method of referenced class.
3. Reference: (Clause A.2)
4. Test Type: Capability.

### A.15.2 Test Geometric complex

1. Test Purpose: test all requirements in the requirement class Geometric complex (Clause 10).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 8.1
4. Test Type: Capability.

## A.16 Conformance class: Topology root: Test

1. Test Purpose: test all requirements in the requirement class Topology root (Clause 11.2).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 11.2.
4. Test Type: Capability.

## A.17 Conformance class: Node

### A.17.1 Dependency Topology root

1. Test Purpose: test Conformance class: Topology root (Clause A.16)
2. Test Method: Use method of referenced class.
3. Reference: Clause A.16
4. Test Type: Capability.


© ISO 2013 – All rights reserved

### A.17.2 Test Node

1. Test Purpose: test all requirements in the requirement class Node (Clause 11.3).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 8.1
4. Test Type: Capability.

## A.18 Conformance class: Edge

### A.18.1 Dependency Node

1. Test Purpose: test Conformance class: Node (Clause A.17)
2. Test Method: Use method of referenced class.
3. Reference: Clause A.17
4. Test Type: Capability.

### A.18.2 Test Edge

1. Test Purpose: test all requirements in the requirement class Edge (Clause 11.4).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 11.4
4. Test Type: Capability.

## A.19 Conformance class: Face

### A.19.1 Dependency Edge

1. Test Purpose: test Conformance class: Edge (Clause A.18)
2. Test Method: Use method of referenced class.
3. Reference Clause A.16
4. Test Type: Capability.

### A.19.2 Test Face

1. Test Purpose: test all requirements in the requirement class Face (Clause 11.5).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 11.5
4. Test Type: Capability.

## A.20 Conformance class: TopoSolid

### A.20.1 Dependency Face

1. Test Purpose: test Conformance class: Geometry (Clause A.16)
2. Test Method: Use method of referenced class.
3. Reference: Clause A.16
4. Test Type: Capability.

### A.20.2 Test TopoSolid

1. Test Purpose: test all requirements in the requirement class Geometric complex (Clause 11.6).
2. Test Method: Inspect the documentation of the application schema or profile.

© ISO 2013 – All rights reserved

3. Reference: Clause 11.6
4. Test Type: Capability.

## A.21 Conformance class: Topological complex

### A.21.1 Dependency Topology root

1. Test Purpose: test Conformance class: Topology root (Clause A.16)
2. Test Method: Use method of referenced class.
3. Reference: Clause A.16
4. Test Type: Capability.

### A.21.2 Test Topological complex

1. Test Purpose: test all requirements in the requirement class Topological complex (Clause 11.7).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 11.7
4. Test Type: Capability.

## A.22 Conformance class: Boundary operators for aggregate objects

### A.22.1 Dependency Topology root

1. Test Purpose: test Conformance class: Topology root (Clause A.16)
2. Test Method: Use method of referenced class.
3. Reference: Clause A.16
4. Test Type: Capability.

### A.22.2 Test Topological complex

1. Test Purpose: test all requirements in the requirement class Boundary operators (Clause 12.2).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 12.2
4. Test Type: Capability.

## A.23 Conformance class: Boolean or set operators

### A.23.1 Dependency Topology root

1. Test Purpose: test Conformance class: Topology root (Clause A.16)
2. Test Method: Use method of referenced class.
3. Reference: Clause A.16
4. Test Type: Capability.

### A.23.2 Test Topological complex

1. Test Purpose: test all requirements in the requirement class Boolean or set operators (Clause 12.3).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference Clause 12.3
4. Test Type: Capability.


© ISO 2013 – All rights reserved

## A.24 Conformance class: Egenhofer operators

### A.24.1 Dependency Topology root

1. Test Purpose: test Conformance class: Topology root (Clause A.16)
2. Test Method: Use method of referenced class.
3. Reference: Clause A.16
4. Test Type: Capability.

### A.24.2 Test Egenhofer operators

1. Test Purpose: test all requirements in the requirement class Egenhofer operators (Clause 12.4).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 12.4
4. Test Type: Capability.

## A.25 Conformance class: Full topological operators

### A.25.1 Dependency Topology root

5. Test Purpose: test Conformance class: Topology root (Clause A.16)
6. Test Method: Use method of referenced class.
7. Reference: Clause A.16
8. Test Type: Capability.

### A.25.2 Test Full topological operators

1. Test Purpose: test all requirements in the requirement class Full topological operators (Clause 12.5).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 12.5
4. Test Type: Capability.

## A.26 Conformance class: Combinations

### A.26.1 Dependency Topology root

1. Test Purpose: test Conformance class: Topology root (Clause A.16)
2. Test Method: Use method of referenced class.
3. Reference: Clause A.16
4. Test Type: Capability.

### A.26.2 Test Topological complex

1. Test Purpose: test all requirements in the requirement class Topological complex (Clause 12.6).
2. Test Method: Inspect the documentation of the application schema or profile.
3. Reference: Clause 12.6
4. Test Type: Capability.

© ISO 2013 – All rights reserved

# Annex B A little spline theory

## B.1 Types of splines

There are essentially two types of splines, fitted curves and approximations.

The fitted curves take some number of constraints (such as passing through the control points, or having particular derivatives) and a class of functions (such as piecewise polynomials with a given number of continuous derivatives) and solve that algebraic problem to produce a parametric curve of the type specified.

Approximations take a fixed function form and use various techniques to return a curve passing "near" the control points. Sometimes this is all that is needed. Usually these approximations have some added property that is useful, such as ease of calculation or some time of transformation invariance (a affine transformation of a Bézier spline is a Bézier spline for its transformed control points, a projective transformation of a NURBS is a NURBS for its transformed control points). Transformation invariance is handy for GI data since it can eliminate the need to do extra work during a coordinate system transformation on geometries defined by these curves.

## B.2 Fitting polynomial spline to curves

Unless otherwise noted, a polynomial spline fits its control points (actually goes through them) at the parameter values given by the knots. For a spline of

- degree n,
- knot list $u_0, u_1, ..., u_p$ (distinct value, multiplicity≡1); and
- coordinate points (DirectPositions) $\vec{P}_0, \vec{P}_1, ..., \vec{P}_p$;

that means that in the interval between any two knots defined by $u_i \le u < u_{i+1}$ (written as "$[u_i, u_{i+1})$"), the curve is n-degree polynomial vector passing between the DirectPositions as defined by $\vec{P}_i$, to $\vec{P}_{i+1}$, with continuity $C^{n-1}$ (i. e. with continuous derivatives of degrees including up to n-1). This means that for each coordinate offset $i$, $1 < i \le n$, there is a piecewise polynomial functions in $p_i(u)$ which are $C^{n-1}$ continuous where the curve defined component-wise by these functions $\vec{C}(u) = [p_1(u), p_2(u), ..., p_n(u)]$ passes through the points in question $0 \le k \le p \Rightarrow \vec{C}(u_k) = \vec{P}_k$. Initial condition on tangent and other derivative vectors (up to the n-1st) can be given at the end points to complete the constraints. These constraints are sufficient to determine the coefficients of each of the polynomial segments in each of the dimensions.

The most common knot sequences are normalized (knot are consecutive integers) or by chord length (each knot spacing equal to the distance between the corresponding poles). Such functions can be computationally intensive, and have few intuitive editors for shape. Each offset being independent makes them totally inappropriate for homogeneous coordinate representations. It also gives these splines some local behavior that is not conducive to coordinate transformations or to projective or affine calculations.


© ISO 2013 – All rights reserved

## B.3 Approximating curves with polynomial splines

A fitted spline can be a lot of calculations, and it is often easier and as affected to use standard weight functions to approximate the poles instead of passing through them. A set of weight functions must be a "partition of unity." This means that within their domain, each function must be non-negative ( $\forall i \in \mathbb{Z} \; \forall \; u \in [0,1] \Rightarrow 0 < B_i(u) \leq 1$ ) the sum of the functions must be a constant one, i.e. $\forall \; u \in [0,1]: \sum_i B_i(u) \; = \; 1$.

Approximating splines use the same polynomial or rational functions for each offset and exhibit invariance under some types of transformations. Bézier and B-splines are invariant of affine transformations, and NURBS (non-uniform rational B-splines) are invariant under projective transformation. That means that if the positions in the control point array are close enough to keep the local curvature tightly bounded along a segment, the transformation of such a spline is nearly the same as the same spline calculated on the transformation of the control points. This in and of itself makes these approximating splines are better behaved than their fitted counterparts under coordinate system transformations.

### B.3.1 Bezier polynomials

The easiest one to understand the Bézier polynomials is an observation derived from the binomial theorem:

$$\forall u \in [0,1] \Rightarrow 1 = 1^p = (u + (1-u))^p = \sum_{i=0}^{p} \binom{p}{i} u^i (1-u)^{p-i}$$

Where

$$\binom{p}{i} = \frac{p!}{i!(p-i)!}$$

So we can define the $i^{th}$ Bézier polynomial of degree p on [0, 1] to be:

$$B_i^p(u) = \binom{p}{i} u^i (1-u)^{p-i}$$

It is trivial to see that these polynomials are all of degree p, are all positive in the interval (0, 1), non-negative in the close interval [0, 1], and always sum exactly to 1.0. If we define a polynomial curve on the sequence of poles (control points) $\vec{P}_0, \vec{P}_1, ..., \vec{P}_p$ as:

$$\vec{C}(u) = \sum_{i=0}^{p} B_i^p(u) \vec{P}_i$$

This curve is defined by a single polynomial function along its entire length. It passes through the first and last pole, and passes "near" the others. The knots on such a Bézier consist of "0" and "1.0." Further, since all the offsets in the poles are treated exactly the same, the formulation is independent of dimension of the coordinate space and works equally well in any dimension or even in homogeneous coordinate representations (more on this later when we discuss rational splines).

© ISO 2013 – All rights reserved

## B.3.2 B-spines and the Basis polynomials

B-Splines are a bit more complex. The polynomials are defined piecewise and recursively on the knot sequence (including multiple degree knots which are repetitions in the sequence). The following formulation of the functions in the partition of unity for B-splines is the called the Cox, de Boors recursion formulae. Given a sequence or real numbers called the knots: $0 \le u_0 \le u_1 \le ... \le u_m \le 1$ with the i[th] knot span defined by: $[u_i, u_{i+1}) = \{u \mid u_i \le u < u_{i+1}\}$

$$N_{i,0}(u) = \begin{cases} 1 & u \in [u_i, u_{i+1}) \\ 0 & u \notin [u_i, u_{i+1}) \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \text{ for } p > 0$$

Note that the roots (0-level) of this recursive tree are functions $N_{i,0}(u)$ which are non-zero on only one knot span. Using them as a set of weight function would result in a "step function" i.e. one which is constant on intervals. The functions in first level of the tree, the $N_{i,1}(u)$, are defined by the sloped and weighted combinations of two roots, and therefore non-zero on two consecutive knot spans or the interval $[u_i, u_{i+2})$. These functions are piecewise linear and if used would result in piecewise linear functions (line strings).The p[th]-level of the tree, the $N_{i,p}(u)$, adds two functions whose non-zero spans are $[u_i, u_{i+p+2})$ and $[u_{i+1}, u_{i+p+3})$. Thus these functions are non-zero on "p+1" knot spans (some of which may be zero-length) comprising the interval $[u_i, u_{i+p+3})$. In all cases, the functions are non-negative; $N_{i,p}(u) \ge 0.0$.

It should be noted that if a "0/0" shows up in the formula above when knots are repeated ("have multiplicity"). In all of these cases, the interval involved is empty (start value = end value).

$$(u_i = u_{i+1}) \Rightarrow ([u_i, u_{i+1}) = \{u \mid u_i \le u < u_{i+1}\} = \{u \mid u_i \le u < u_i\} = \varnothing).$$

Since the condition on the interval would never be satisfied, and so the equation would never be actually used. To keep this "undefined" issue from arising, the texts usually make the assumption that "0/0 = 1.0" which is wildly invalid in any other situation.

These functions, called the "B-spline basis functions" are used as partitions of unity and given a set of (n+1) $\vec{P}_0, \vec{P}_1, ..., \vec{P}_n$ control points, a knot vector of length (m+1) values (some possibly repeated) $u_0 \le u_1 \le ... \le u_m$, the B-spine of degree p (where $m = n + p + 1; p = m - n - 1$) is defined as:

$$Poles = \{\vec{P}_0, \vec{P}_1, ..., \vec{P}_n \in DirectPosition\}$$

$$Knots = \{u_i \mid u_i \in \mathbb{R}, u_0 \le u_1 \le ... \le u_m\}$$

$$\vec{C}(u) = \sum_{i=0}^{p} N_{i,p}(u) \vec{P}_i$$

A B-Spline curve is "clamped" if it is forced through the first and last control points. This is done by making the first and last knot have multiplicity "$p+1$" which forces the curve to start at $\vec{P}_0$ and to


© ISO 2013 – All rights reserved

end at $\vec{P}_n$. Generally speaking, B-splines need to be clamped to maintain the topology of lines and their start and end points (normally the first and last control point). Implementation can chose to not do this, but it is difficult to do and maintain topological consistency.

A B-Spline surface is "clamped" if each of its horizontal (u – varying, v – fixed) and vertical (u – fixed, v – varying) curves are clamped. This means that the knots for each variable, at each end of the sequence have multiplicity "$p+1$."

A B-spline can be made to close back on itself by a repetition of the first and last "$p$" poles, and knot spacing.

## B.4 Surface splines

Spline surfaces are created by "tensor" curve splines. For example, suppose we have a 2-dimensional array of poles, $\{\vec{P}_{i,j} \mid 0 \le i \le n; 0 \le j \le m\} \subset DirectPositions$, and 2 1-dimensional arrays of knot ($u_0 \le u_1 \le ... \le u_n$ and $v_0 \le v_1 \le ... \le v_m$). Then we can create a spline curve of spline curves as:

$$s_{p,q}(u,v) = \sum_{i=0}^{p} \sum_{j=0}^{q} N_{i,p}(u) N_{j,q}(v) \overrightarrow{P_{i.j}}$$

This can be thought of as a 1-parameter set of curves, and if the poles are spread out sufficiently, the topology of the image of this 2D function is a 2D geometry, a surface. The example here is for B-splines, but similar formula will work for any tensor of 2 spline curve formulations. For this international standard, the usual practice is to use like-on-like, i.e. 2D B-spline or 2D polynomial, and avoid a 2D mixed tensors like Bézier s on B-splines.

## B.5 Rational splines and homogeneous coordinates

Like going from curves to surfaces, the manner of going from approximated polynomial splines to approximated rational splines is a "simple" algebraic trick. Like most simple things, it is only simple after you see and understand it.

Recall that in the definition of DirectPositions in the standard, we do not distinguish between normal Cartesian and projective space coordinates, this means that we do not distinguish between the following "tuples of numbers: "

$$\forall x, y, z, w \in \mathbb{R}; w \ne 0$$
$$(x, y, z) \leftrightarrow (x, y, z, 1) \leftrightarrow (wx, wy, wz, w)$$

The first tuple is a 3D coordinate point in normal Euclidean form. The second tuple is a 3D coordinate point lying on a plane $w=1$ in 4D (x, y, z, w), which maps to and from the first tuple. The last tuple, if $w \ne 0$, is a point on a line in 4D from the origin through the point in the second tuple, except that in this case, it is on the plane defined by the parameter $w$.

Now the trick; suppose we have a sequence of poles $\vec{P}_0$, $\vec{P}_1$, ..., $\vec{P}_n$ in 3D Euclidean space, and we have an equi-length sequence of weights $w_0$, $w_1$, ..., $w_n$, we can then define our equivalence by:

© ISO 2013 – All rights reserved

$$(x,y,z) \xrightarrow{w} (wx,wy,wz,w)$$
$$\forall i = 0,1,...,m$$
$$\vec{P}_i \xrightarrow{w_i} \vec{H}_i$$

It takes a little algebra to see but a spline (here a B-spline) on the $\{\vec{H}_i\}$ can similarly mapped to another (B-spline) spline on the $\{\vec{P}_i\}$. And the mapping works from a degree p spline in projective space (homogeneous coordinates) to one in Euclidean space works this way:

$$s^h(u) = \sum_{i=0}^{n} N_{i,p}(u)\overrightarrow{H_i}$$
$$s^e(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i \overrightarrow{P_i}}{\sum_{i=0}^{n} N_{i,p}(u) w_i}$$

Note: The above "trick" means that any approximating spline functionality working on vectors as DirectPositions can be used to implement the corresponding "rational function version" of that spline algorithm by use of homogeneous coordinates.

For surfaces the corresponding equations (degree p horizontal and q vertical) are:

$$s^h(u) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v)\overrightarrow{H_{i.j}}$$
$$s^e(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{i,j} \overrightarrow{P_{i.j}}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{i,j}}$$


© ISO 2013 – All rights reserved

# Annex C (informative) Examples of spatial schema concept

## C.1 GeometrySemantics

The examples here use the names of the types in the normative part of this document as if they were instantiable classes. While not the normal UML semantics for type, this mnemonic is justifiable under several interpretations. First, the conformance clause does not require that the types in this International Standard be included in an application schema, but that classes in the application schema realize these types. This logical requirement does not require the instantiated classes to be named differently from the standard's types and interfaces. Second, assuming a design system that uses a strong name space convention, items in different name spaces can have the same local name. In other words, local names are not globally unique. Third, the examples are valid for any implementation classes that realize the types so identified. Any implementation (application schema) would have to have a schema map that associates these types with implementation classes that realize them. The proper use of that map would result in valid syntax.

In general, it is valid to use common names for "metaphorically identical" but technically different entities. The UML model in this International Standard defines abstract types, application schemas define conceptual classes, various software systems define implementation classes or data structures, and the XML from the encoding standard defines entity tags. All of these reference the same information content. There is no difficulty in allowing the use of the same name to represent the same information content even though at a deeper level there are significant technical differences in the digital entities being implemented. This "allows" types defined in the UML model to be used directly in application schemas.

## C.2 Geometric objects in a 2-dimensional coordinate reference system

This example is based on a simple decoding scenario. This is used as opposed to an editing use case because it eliminates the need to discuss the fine points of creating a viable topology editor. The following assumptions are made about the application schema (defined in accordance with Rules for application schema):

The geometry and topology schema are compliant with the spatial schema defined in this document, and therefore include instantiable subclasses of the major geometry and topology types defined in the normative part of this document. For the sake of readability, the type names used in the normative part of this document are used in lieu of their instantiable subtypes.

The schema includes the requirement to use a full planar topology.

The schema includes a 2D coordinate reference system.

The feature schema includes the equivalent of theme, feature and feature components as described in the discussion of MiniTopo in Annex D.

Persistent objects, after creation, are inserted into a datastore called "Datastore".

Figure C.1 represents the geometry of a GeometryComplex, based on a planar manifold. To construct this complex, the following example uses a functional cascade, where objects are created with constructors based on the coordinates given in the diagram. Once an object has been created it can be used in any subsequent formulation. For objects not given formal constructors in the normative section, a default one is assumed that simply takes a record representation of the state of the object

© ISO 2013 – All rights reserved

and uses it as a parameter to a data type-like constructor. This is very consistent with how this would be done in SQL 99. SQL automatically creates default constructors for any UDT (user defined type) based on the requirements of an insert semantics. Recall that "< >" denotes a record, or an ordered set (list), and that "{ }" denotes an unordered set or bag.

Construction can begin with the creation of the points. There is a minor issue here since Point, being a type, cannot be instantiated. To be a compliant application schema, a instantiable class that is a subtype of Point must be included, and this class would have to be substituted in the creation cascade below for each use of Point. First, the 7 Points, indicated by dots and identified as {P1, . . . P7} are created:

```
P1=Point < position=< 1.00, 5.00 > >
P2=Point < position=< 3.00, 5.00 > >
P3=Point < position=< 3.00, 2.00 > >
P4=Point < position=< 1.75, 2.75 > >
P5=Point < position=< 1.50, 4.50 > >
P6=Point < position=< 2.00, 3.25 > >
P7=Point < position=< 5.00, 4.00 > >
Insert P1, P2, P3, P4, P5, P6, P7 into Datastore
```

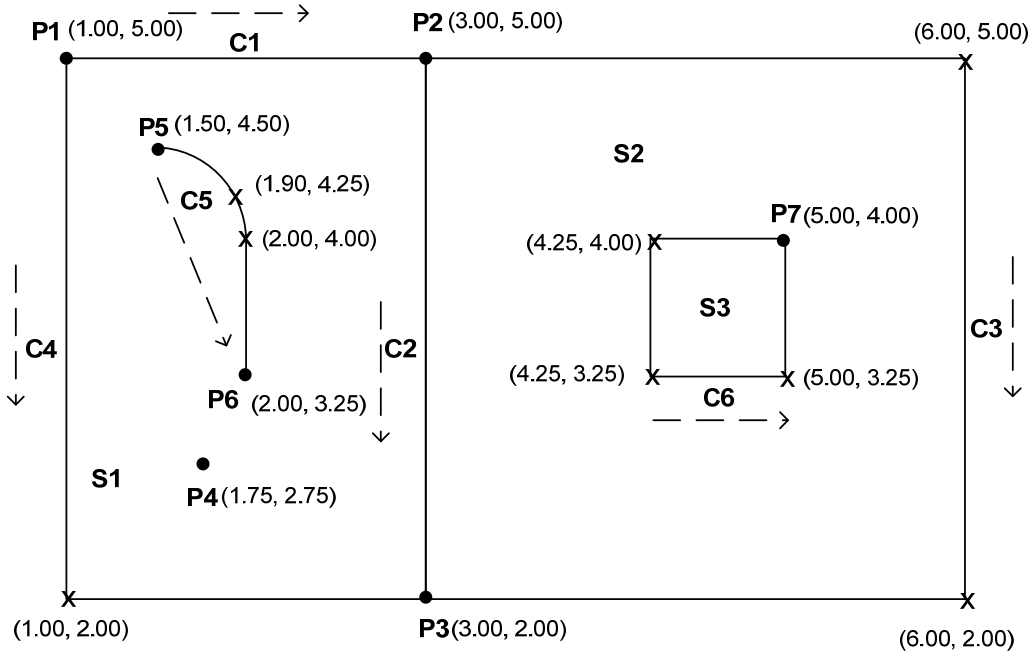



**Figure C.1 — A data set composed of the GeometryPrimitives**

With the existence of the points, the cascade can continue with the construction of the 7 Curves, identified {CS1, CS2, CS3, CS4, CS5, CS6, CS7} which can be used to construct the curves to follow. Recall that subtypes of Curve are data types and cannot hold persistent identification. Thus, the variables used to define the curve segments below are "heap" or local variables, defined within the context of the construction, but not persistently stored until they are included as members of an object type (in this case, the curves defined later). All of the curve segments defined here are either line strings or arcs.

```
CS1=Curve <controlPoint=<P1,P2>, interpolation="linear" >
```


© ISO 2013 – All rights reserved

```
CS2=Curve <controlPoint=<P2,P3 >, interpolation="linear" >
CS3=Curve <controlPoint=<P2,(6,5),(6,2),P3>,
interpolation="linear" >
CS4=Curve <controlPoint=<P1,(1,2), P3> ,
interpolation="linear" >
CS5=Curve <controlPoint=<P5,(1.9,4.25),(2,4)>
interpolation="arc">
CS6=Curve <controlPoint=<(2,4),P6>, interpolation="linear" >
CS7=Curve <controlPoint=<P7,(4.25,4),(4.25,3.25),(5,3.25),P7 >,
interpolation="linear">
```

There is a hidden assumption here that the persistent variables, such as P1, which have previously been entered into the datastore can be accessed so that the local copy and persistent copy are maintained in synchrony. This allows the insertion of the curve segments (as members of the curves below) to proceed while still using the Point variant of the DirectPosition data type. In an object relational database scenario using only an SQL language application program interface (API), the application would track references to variables and use them in subsequent insert statements. In a similar scenario using an object interface to the same datastore, the database API would make this tracking issue transparent to the programmer.

The curve segments can now be used to construct persistent objects: 6 Curves, identified as {C1, . . . C6}. The same comment about instantiable types applies, in that the local application schemas' required subtype of Curve would have to be used instead of Curve.

```
C1=Curve segments=<CS1>
C2=Curve segments=<CS2>
C3=Curve segments=<CS3>
C4=Curve segments=<CS4>
C5=Curve segments=<CS5, CS6>
C6=Curve segments=<CS7>
Insert C1, C2, C3, C4, C5, C6 into Datastore
```

The curves can then be used in the construction of surfaces. In this case, the planar polygon constructor can be used, since our coordinate space is 2D. The upNormal of the surfaces is the standard upNormal of the surface (often denoted as *k*), and need not be specified. Since the intent is to define a full topology complex, we need a complete coverage by surfaces of the area of the coordinate surfaces. Since the universal face is often referred to as "Face 0", we define here a S0 to be the geometric realization of that face. Thus, the 4 Surfaces are identified as {S0, S1, S2, S3}.

```
S0=Surface patch=<Polygon interior=<< C1, C3, -C4 >> >
•    this universal face is only needed to construct a
     topological complex
•    with a full planar graph
S1=Surface patch=<Polygon exterior=< C4, -C2, -C1 >,
                          interior=<< C5, -C5 >> >
S2=Surface patch=<Polygon exterior=< -C3, C2 >,
                          interior=<< -C6 >> >
S3=Surface patch=<Polygon exterior=< C6 > >
Insert S0, S1, S2, S3 into Datastore
```

© ISO 2013 – All rights reserved

All the necessary pieces of geometry exist for the creation of a GeometryComplex, which is a type of GeometryObject collection, it is necessary only to give an exhaustive list of the required objects. This can cascade directly into creation of a Complex.

```
GComplex=GeometryComplex < surfaces  ={S0, S1, S2, S3},
                   curves  ={C1, C2, C3, C4, C5, C6}
                   points  ={P1, P2, P3, P4, P5, P6, P7} >
TComplex=Complex <  realization=GComplex >
Insert GComplex, TComplex into Datastore
```

This concludes the geometric constructions describing the geometry and topology in the diagram at the beginning of this clause. Although out of the control of this document, the construction of features (Figure C.2) might conclude this scenario as follows:

```
Lake    = AreaFeature featureType="Hydrography::WaterBody",
     extent=S3
RoadCenterline=  LineFeature featureType="Transportation::Road",
               centerline=C2
RoadArea  = RoadCenterLine.centerline.buffer < distance=10m >
RoadExtent   = AreaFeature featureType="LandCover::Road"
               extent=RoadArea
RoadInstance=ComplexFeature featureType="LandUse::Road",
               featureComponents={RoadCenterline, RoadArea }
Trail     = LineFeature
     featureType="CulturalFacilities::HikingTrail",
               centerline=C5
School    =PointFeature  featureType="CulturalFacilities::School",
               Location=P4
Insert Lake, RoadCenterline, RoadExtent, RoadInstance, Trail,
     School
     into Datastore
```


© ISO 2013 – All rights reserved

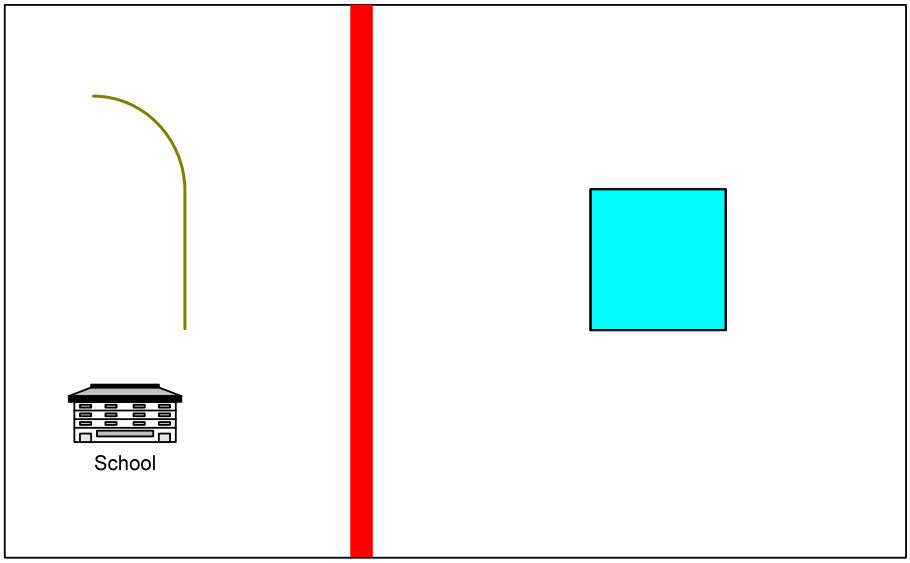



**Figure C.2 — Simple cartographic representation of sample data**

## C.3 Geometric objects in a 3-dimensional coordinate reference system

In Figure C.3, we have a 3D solid with planar facets. It is a rectangular block into which has been cut a rectangular slot, which is counter sunk by one unit.

```
     P1=Point position=<2.00, 5.00, 4.00>
P2=Point position=<5.00, 5.00, 4.00>
P3=Point position=<5.00, 3.00, 4.00>
P4=Point position=<2.00, 3.00, 4.00>
P5=Point position=<2.00, 5.00, 2.00>
P6=Point position=<5.00, 5.00, 2.00>
P7=Point position=<5.00, 3.00, 2.00>
P8=Point position=<2.00, 3.00, 2.00>
P9=Point position=<1.00, 5.00, 1.00>
P10=Point position=<9.00, 5.00, 1.00>
P11=Point position=<9.00, 1.00, 1.00>
P12=Point position=<1.00, 1.00, 1.00>
P13=Point position=<1.00, 5.00, 7.00>
P14=Point position=<9.00, 5.00, 7.00>
P15=Point position=<9.00, 1.00, 7.00>
P16=Point position=<1.00, 1.00, 7.00>
```

© ISO 2013 – All rights reserved

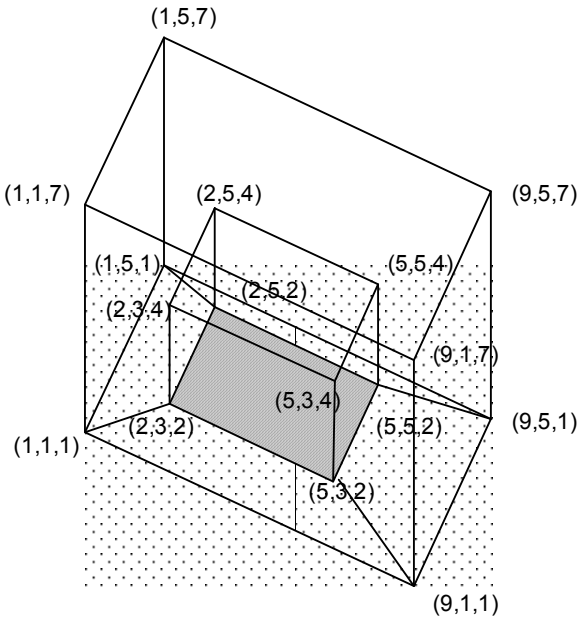



**Figure C.3 — A 3D Geometric object with labeled coordinates**

The surface can be expressed as a GriddedSurface (wrapped around on itself to make a topological cylinder) and 2 Polygons (to act as end caps for the topological cylinder), all with planar interpolations.

```
S1=Surface patch=
< <BilinearGrid rows=4, columns=5,
     controlPoint=< <P1, P2, P3, P4, P1>,
                    <P5, P6, P7, P8, P5>
                    <P9, P10,P11,P12,P9>,
                    <P13,P14,P15,P16,P13> >  ,
     Polygon exteriorVertices=<P1, P2, P3, P4, P1 >,
     Polygon exteriorVertices=<P16,P15,P14,P13,P16> >
```

The example in Figure C.4 consists of a Point [P1], a Curve [C1], and a Surface [S1]. The segmentation association of the Surface points to 9 SurfacePatches. The first SurfacePatch represents the area to the left of the dashed line. The other 8 SurfacePatches, all Triangles, represent the area to the right of the dashed line.


© ISO 2013 – All rights reserved

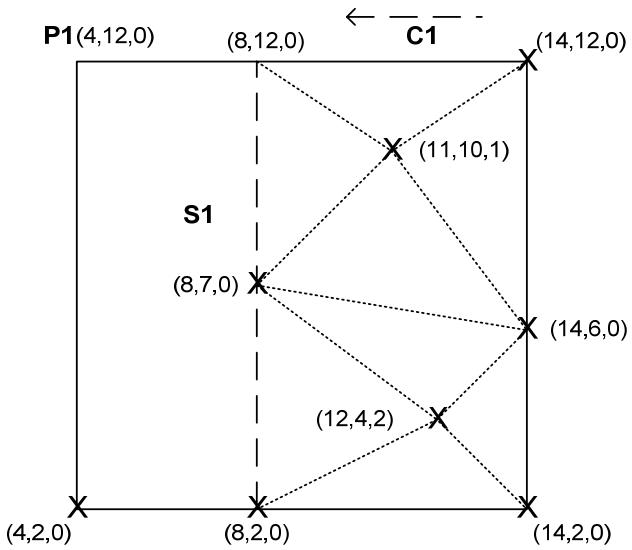



**Figure C.4 — Surface example**

```
P1=Point(4,12,0)
C1=Curve segment=<Segment 1>
Segment 1=Curve
     controlPoint=<(4,12,0),(4,2,0),(14,2,0),(14,12,0),(4,12,0)>
Patch1=Polygon exterior=<P1,(4,2,0),(8,2,0),(8,12,0), P1>
Post1=DirectPosition(8,12,0)
Post2=DirectPosition(14,12,0)
Post3=DirectPosition(11,10,1)
Post4=DirectPosition(8,7,0)
Post5=DirectPosition(14,6,0)
Post6=DirectPosition(12,4,2)
Post7=DirectPosition(8,2,0)
Post8=DirectPosition(14,2,0)
T1=Triangle exterior=<Post1, Post2, Post3, Post1>
T2=Triangle exterior=<Post1, Post3, Post4, Post1>
T3=Triangle exterior=<Post3, Post5, Post4, Post3>
T4=Triangle exterior=<Post2, Post5, Post3, Post2>
T5=Triangle exterior=<Post4, Post5, Post6, Post4>
T6=Triangle exterior=<Post4, Post6, Post7, Post4>
T7=Triangle exterior=<Post5, Post8, Post6, Post5>
T8=Triangle exterior=<Post7, Post6, Post8, Post7>
S1=Surface patch=<Patch1, T1, T2, T3, T4, T5, T6, T7, T8>
```

Note that the same example could be described as a set of two Surfaces, one composed of a single SurfacePatch, P1, and the other, a TriangulatedSurface composed of the eight Triangles. Those two Surfaces could then be combined into a CompositeSurface equivalent to the single Surface described above.

© ISO 2013 – All rights reserved