GEOGRAPHY MARKUP LANGUAGE

David S. Burggraf*

** Galdos Systems Inc., Vancouver B.C., V6C 1T2*
Email: dburggraf@galdosinc.com

ABSTRACT

Geography Markup Language (GML) is an XML application that provides a standard way to represent geographic information. GML is developed and maintained by the Open Geospatial Consortium (OGC), which is an international consortium consisting of more than 250 members from industry, government, and university departments. Many of the conceptual models described in the ISO 19100 series of geomatics standards have been implemented in GML, and it is itself en route to becoming an ISO Standard (TC/211 CD 19136). An overview of GML together with its implications for the geospatial web is given in this paper.

1 INTRODUCTION

A Geographic Information System (GIS) is used for all sorts of decision making in various domains ranging from navigation to transportation management to emergency response. The first operational Geographic Information System was developed in the mid 1960's in Ottawa, Ontario, which became known as Canadian GIS or CGIS. Two decades later, several digital storage formats were available to encode geographic information. Today most geographic data sets are encoded in disparate formats and models and typically reside in a wide variety of unconnected databases. In the cases where geographic information is shared, conflicts or discrepancies between one data set and another are usually not detected, such as at the boundaries between municipalities or the path of a road from one state to another. Problems like storms, floods, or forest fires are rarely contained within a single jurisdictional boundary, and this makes it difficult to combine the information as needed to respond to a given problem. Geography Markup Language (GML) was designed to solve such problems.

Geography Markup Language (GML) is an XML application that provides a grammar and base vocabulary for describing geographic data. GML was developed in order to provide a standard means of representing information about geospatial features—their properties, interrelationships, and so on. Features describe real world entities and are the fundamental objects in GML. Features can be concrete and tangible, such as roads and buildings, or abstract and conceptual, such as political boundaries and distributions of quantities over geographical areas (coverages). GML features are described in terms of their properties, which can represent geometric, topological and temporal characteristics or associations with other features. For instance, GML can describe the location, shape, and extent of geographic objects as well as properties such as colour, speed, and density, some of which may depend on time. One could describe a natural disaster, such as a flood or storm front as a dynamic feature in GML, whose properties such as extent, water temperature, and expansion rate are all recorded at various time slices. As it is impossible to describe all features and predict their usage a priori, the GML core schemas do not fix definitions of specific instantiable feature types such as a road or bridge. Rather, specific features are defined in GML Application Schemas, which are created by users.

The GML specification is developed and maintained by the Open Geospatial Consortium (OGC), an international consortium of more than 250 companies, government agencies, and university departments participating in a consensus process to develop publicly available geo-processing specifications. GML 3.0 implemented many of the conceptual models described in the ISO 191XX series of geomatics standards and was formally adopted by the OGC in January 2003. The current version is 3.1.1, which is en route to becoming an ISO Standard (TC/211 CD 19136).

GML together with the Open Geospatial Consortium's (OGC) vendor-neutral web services enable existing geographic databases to be interconnected in a similar way as the things in the world are interconnected. GML plays a central role in the information infrastructure for a web of geographic information integrating municipalities, private sector corporations, utilities, and all levels of government. This sets the stage for the *Geo-web*—an evolving network of globally accessible and locally maintained databases containing current geographic information.

2 GML FEATURES

GML is based on a feature framework. Domain experts create specific objects that make up the vocabulary for the domain by deriving from the appropriate GML core objects. Such domain specific objects might include, for example, roads, rivers, faults, boundaries, sensors, etc. Features in GML are represented by XML elements, with namespace qualified names (e.g. <abc:Road>) that are described by their child property elements. Feature properties can have spatial and temporal values, which can be static or dynamic.

One of the primary objectives of GML is to provide a language for expressing geographic features in a manner that is shareable over the Internet. As a result, GML imposes additional structure on XML such as syntactic rules regarding the use of attributes and the classification of elements into two groups—objects and properties. GML provides a set of core schema components such as abstract feature objects and other supporting objects (e.g. geometry, topology, temporal) together with a simple semantic model between objects and properties that is similar to the Unified Modeling Language (UML) class/association model, the Entity-Relationship (ER) model, and the Resource Description Framework (RDF) subject/property model.

Using the GML object/property model and its schema components, users can describe the geographic types, whether concrete or conceptual, which are used within their application domain. The set of objects is created in the form of one or more shareable GML application schemas, which are XML schemas that import GML schema components and comply with the GML semantic model and syntactic rules. A key benefit of GML is that the application schemas can be published, extended, and shared over the Internet, as shown in Figure 1, which is critical to any regional, national, or international information infrastructure. GML as an XML application is ideally suited for this purpose as opposed to UML for which such sharing is not possible in any meaningful way.

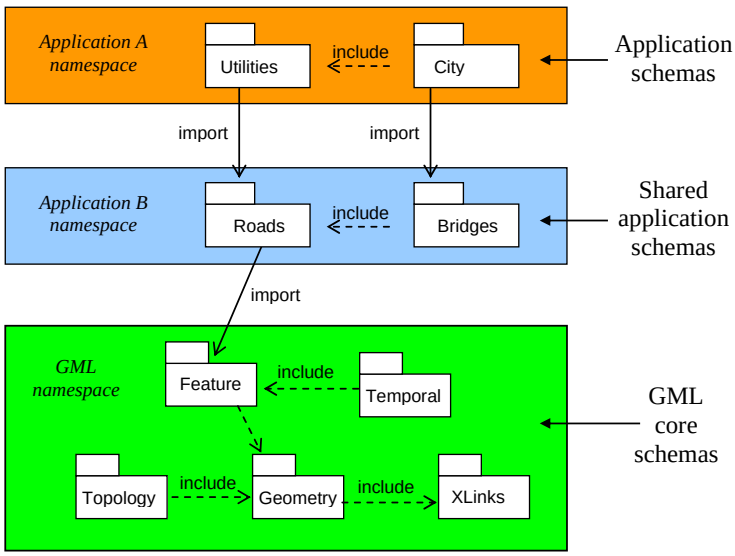

Figure 1. Every GML Application Schema must import (at least transitively) one or more GML core schemas.

2.1 The Object/property model

The GML object is encoded in XML by declaring a global element together with a corresponding complex type and then defining child elements of that type. These child elements are called properties of the object.

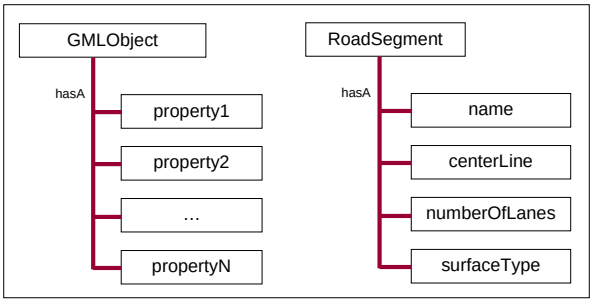

Figure 2. A GML Object is described by its property children.

Figure 2 reveals a GML syntactic convention used to distinguish between objects and properties; element and type names representing GML objects are written in UpperCamelCase and the property names are written in lowerCamelCase.

In contrast to legacy GIS approaches, a Feature is not defined primarily as a geometric object but as a meaningful object that might have some properties that are geometric and other properties that are not. For example, Figure 2 shows that a RoadSegment feature has three properties that describe its physical characteristics and one property that describes its name. Properties can be used to describe relationships (associations) between two GML objects, where the property name should provide the name of the relationship or possibly the name of the role that the target (or source) value plays in the relationship. This is illustrated in Figure 3.

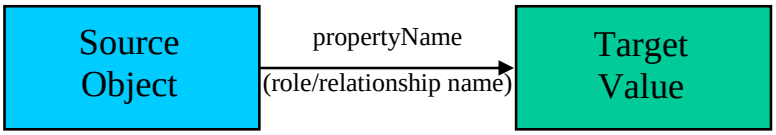

Figure 3. A property name in GML indicates the name of the relationship with or role of the target value.

Figure 2 shows that the RoadSegment has a centerLine property. Suppose the value of the centerline property is a geometry object such as a LineString (piecewise linear curve). The property name centerLine in this case represents the role that the target value plays with respect to the source value in the relationship and is illustrated in Figure 4. Note that it is possible to describe several different relationships between the RoadSegment and other geometries. For example, the RoadSegment could have another property named extent whose value is a Polygon, which would represent its surface extent, and a property called center that is Point valued, which would represent the location of the center of its surface extent.

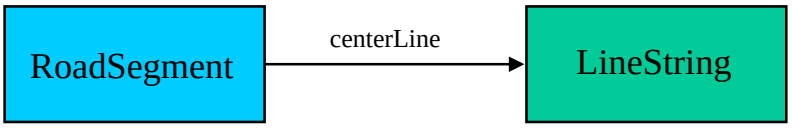

Figure 4. The relationship between a feature and a geometry object is expressed by the property centerLine.

A GML object cannot directly have second GML object as a child element; rather a child property element must be defined whose value is the second object. The following example shows a correct sample instance of the RoadSegment feature in GML:

```
<app:RoadSegment gml:id = "RS001">
    <app:name>Hanbury Road North</app:name>
    <gml:centerLine>
        <gml:LineString gml:id="L001">
            <gml:coordinates>1,2 2,3 3,4 4,0</gml:coordinates>
        </gml:LineString>
    </gml:centerLine>
    <app:numberOfLanes>2</gml:numberOfLanes>
    <app:surfaceType>Asphalt</gml:surfaceType>
</app:RoadSegment>
```

In this example, app:RoadSegment refers to the RoadSegment feature as defined in an application schema that resides in the application namespace. The prefix "app" is an arbitrary string that represents the application namespace and serves only as a place-holder for the application namespace. The RoadSegment instance is uniquely identified by the attribute gml:id.

2.2 Remote Properties

The values of GML properties can be defined either inline or remotely. In the previous RoadSegment sample instance, all of the property values were defined inline. Remote property values are encoded as shown in the following example:

```
<app:RoadSegment gml:id = "RS1">
    <app:name>Hanbury Road North</app:name>
    <gml:centerLine xlink:href="#L001"/>
    <app:numberOfLanes>2</gml:numberOfLanes>
    <app:surfaceType>Asphalt</gml:surfaceType>
</app:RoadSegment>
```

The centerLine property references a LineString geometry defined elsewhere by using an xlink:href attribute. In this case the xlink:href attribute, from the XLinks namespace, refers to a GML type in the same document whose unique identifier is gml:id="L001". In general, remote property values can reference a GML object located anywhere on the Internet.

2.3 The Role of Attributes in GML

In GML instances, properties of features and other objects are expressed by elements not attributes. For example, the following grammar is not valid GML:

Although it is not invalid in XML schema to use the attributes as shown above, GML aware software will not interpret these attributes as properties of the GML feature. In GML instances, only a few attributes such as gml:id, srsName, and xlink:href are used, and these are used only as supportive constructs, rather than carrying object properties.

2.4 Feature Relationships

A property is used to express a relationship between two features, where the property name provides the name of the relationship. For example, in the case where a Road crosses a Creek the relationship between the Road and Creek is encoded as follows.

```
<app:Road gml:id = "Rd001">
    <gml:name>Robert's Creek Road</gml:name>
    <app:crosses>
        <app:Creek gml:id = "Cr001">
            <gml:name>Robert's Creek</gml:name>
        </app:Creek>
    </app:crosses>
</app:Road>
```

The crosses property name in this case provides the name of the role that the source (Road) feature plays in the relationship. Since the Creek is crossed by the Road the inverse relationship can expressed by a crossedBy property element as shown in the following instance.

```
<app:Road gml:id = "Rd001">
    <gml:name>Robert's Creek Road</gml:name>
    <app:crosses>
        <app:Creek gml:id = "Cr001">
            <gml:name>Robert's Creek</gml:name>
            <app:crossedBy xlink:href = "#Rd001"/>
        </app:Creek>
    </app:crosses>
</app:Road>
```

Sometimes, it is preferable to use a single property name to represent a bi-directional relationship. The intersects property is one such example as shown in the following instance:

```
<app:Road gml:id = "Rd001">
    <gml:name>Robert's Creek Road</gml:name>
    <app:intersects>
        <app:Creek gml:id = "Cr001">
            <gml:name>Robert's Creek</gml:name>
            <app:intersects xlink:href = "#Rd001"/>
        </app:Creek>
    </app:intersects>
</app:Road>
```

3 GML CORE COMPONENTS

Many new components were added to GML since version 2.X. There were only 3 core schemas in GML2, namely feature.xsd, geometry.xsd, and xlinks.xsd, and this grew to twenty-eight core schemas in GML3.0. The model has remained the same respecting features and their properties; however, since version 3.0, GML supports many more objects including topology, temporal components, dynamic features, coverages, and coordinate reference systems. This section provides an overview of these objects.

3.1 Geometry Objects

GML supports a three-dimensional geometry model consisting of objects that represent points, curves, surfaces, and solids. The geometry primitives supported in GML2 were Point, LineString, LinearRing, Box, and Polygon, as well as the corresponding aggregates: MultiPoint, MultiLineString, MultiPolygon. GML now supports

many new types in the geometry schemas including: Arc, Circle, CubicSpline, Ring, OrientableCurve, OrientableSurface, and Solid. In addition to the aggregates such as MultiCurve, MultiSurface, etc., GML now supports the following composites: CompositeCurve, CompositeSurface, and CompositeSolid. The main difference between a geometry aggregate and a composite in GML is that the geometry members of a composite are all connected; whereas the members of an aggregate can be disjointed. Note that in GML, composite is not to be confused with a composition association from UML. The geometry members of a composite do not "live or die" with the composite; they can exist independently.

Rather than having all the geometry types and elements in GML lumped into one schema, the different types are factored into the following five schemas:

1. geometryBasic0d1d.xsd
2. geometryBasic2d.xsd
3. geometryAggregates.xsd
4. geometryPrimitives.xsd
5. geometryComplex.xsd

The first three schemas contain the most commonly used geometry components (the linear geometries) and are backwards compatible with GML 2. The last two geometry schemas contain most of the new types and elements including many non-linear geometries.

3.1.1 Interpolated Curves

The most commonly used one-dimensional geometry type in GML is the LineString. A LineString in GML is a curve that is linearly interpolated between a finite set of points called control points or vertices.

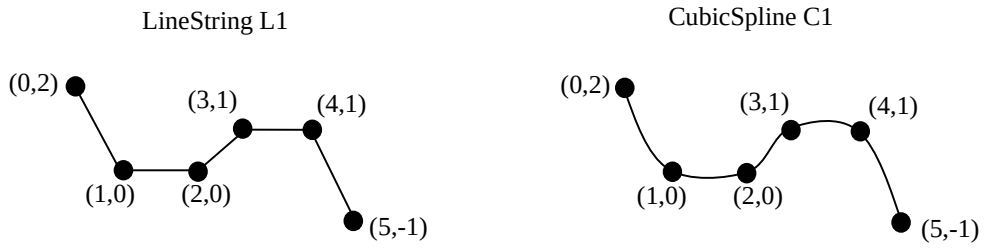

Figure 5. Linear versus non-linear geometries. A (piecewise) linear curve can be represented by a LineString (left). A non-linear CubicSpline curve is shown on the right.

The LineString in Figure 5 can be encoded in GML as follows:

```
<gml:LineString gml:id="L1">
    <gml:pos>0 2</pos>
    <gml:pos>1 0</pos>
    <gml:pos>2 0</pos>
    <gml:pos>3 1</pos>
    <gml:pos>4 1</pos>
    <gml:pos>5 -1</pos>
</gml:LineString>
```

Note that each control point in the encoding above is recorded in a <pos> element, which is of gml:DirectPositionType. The control points of L1 can alternatively be encoded more compactly as "coordinate tuples" using the <coordinates> element as follows:

```
<gml:LineString gml:id="L1">
    <gml:coordinates decimal="." cs="," ts="&#x20;">
        0,2 1,0 2,0 3,1 4,1 5,-1
    </gml:coordinates>
</gml:LineString>
```

Note that the attributes decimal, cs and ts are used to format the decimal, coordinate separator, and tuple separator, respectively. In this case, the tuple separator is set to a single white space by specifying its hexadecimal code: ts=" ". In the case that the attribute values of decimal, cs and ts are not specified, the default values are ".", ",", and " ", respectively. GML supports other non-linear interpolation methods for curves including circular, elliptical, and conic interpolation methods. GML also supports a well-known family of interpolated CurveSegments called Bsplines, of which Bezier polynomial splines and CubicSplines are particular examples. A gml:CurveSegment has a uniform interpolation type and always occurs as a segment of a gml:Curve. A gml:Curve has a segments property whose value is an array of gml:CurveSegments. The cubic spline curve C1 in Figure 5 is encoded in GML as follows:

```
<gml:Curve gml:id="C1">
    <gml:segments>
        <gml:CubicSpline>
            <gml:coordinates>0,2 1,0 2,0 3,1 4,1 5,-1</gml:coordinates>
            <gml:vectorAtStart>1 -3</gml:vectorAtStart>
            <gml:vectorAtEnd>1 -3</gml:vectorAtEnd>
        </gml:CubicSpline>
    <gml:segments>
<gml:Curve>
```

Notice that in addition to providing the control points of the CubicSpline, the vectors encoded in the vectorAtStart and vectorAtEnd properties are necessary to uniquely determine the cubic spline going through the given set of control points.

Furthermore, note that the curves L1 and C1 have an inherent direction that is determined by the order of the control points. A Curve that traverses C1 in the opposite direction—from (5,-1) to (0,2)—can be defined using gml:OrientableCurve and is encoded as follows:

```
<gml:OrientableCurve gml:id="C2" orientation="-">
    <gml:baseCurve xlink:href="#C1"/>
</gml:OrientableCurve>
```

GML supports other non-linear curve segments such as Arc, ArcString and Circle. A gml:Arc is just a connected subset of a circle that is encoded using circular interpolation between three control points. For example, the semicircle that passes through the points (1,0), (0,1), (-1,0) would be encoded as follows:

```
<gml:Arc>
    <gml:coordinates>1,0 0,1 –1,0</gml:coordinates>
</gml:Arc>
```

An ArcString consists of one or more Arcs that are contiguous (connected at endpoints) and a Circle is an Arc that closes up to form a loop. Sample encodings of an ArcString and a Circle are as follows:

```
<gml:ArcString>
    <gml:coordinates>1,0 0,1 –1,0 –2,-1 –3,0</gml:coordinates>
</gml:ArcString>

<gml:Circle>
    <gml:coordinates>1,0 0,1 -1,0</gml:coordinates>
</gml:Circle>
```

Notice that the number of control points in an ArcString must be an odd integer greater than or equal to three—the first Arc uses three, and each additional Arc uses two more.

3.1.2 Surfaces, Surface Patches, and Solids

Surfaces and solids have also been supported since GML 3.0. The boundary of a gml:Solid is a gml:CompositeSurface, which consists of one or more surface members that enclose the Solid without overlap or open gaps. Figure 6 shows an example of a solid half-ball S1 having a composite surface as its boundary consisting of two surfaces, the lower hemisphere H1 and the equatorial plane P1.

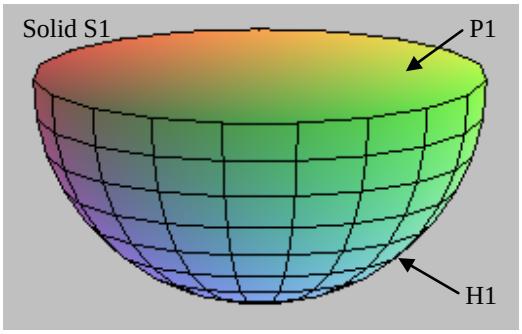

Figure 6. A Solid bounded by a CompositeSurface

A gml:Surface such as the lower hemisphere H1 is encoded as a collection of surface patches, where each patch member is usually a small (flat) polygon. In this case, the surface looks like the surface of a cut gem. However, a more accurate representation of H1 can be defined in an application schema using spherical surface interpolation. An example of this is given in the next section. The following example shows a sample encoding of the Solid S1. Note that a collection of small polygonal patches are used to represent the upper hemispherical Surface H1:

```
<gml:Solid gml:id="S1">
    <gml:exterior>
        <gml:CompositeSurface>
    <gml:surfaceMember>
        <gml:Surface gml:id="H1">
            <gml:patches>
                <gml:PolygonPatch>
                <gml:exterior>
                <gml:LinearRing>
                    <gml:coordinates>
                        1.0,0.0,0.0 0.95,0.31,0.0 0.94,0.30,-0.14 0.94,0.0,-0.14 1.0,0.0,0.0
                    </gml:coordinates>
                </gml:LinearRing>
                    </gml:exterior>
                </gml:PolygonPatch>
            <gml:PolygonPatch>
                ...
            </gml:PolygonPatch>
            ...
                    </gml:patches>
                </gml:Surface>
        </gml:surfaceMember>
        <gml:surfaceMember>
                <gml:Surface gml:id="P1">
                <gml:patches>
```

```
                <gml:PolygonPatch>
                    <gml:exterior>
                        <gml:LinearRing>
                            <gml:coordinates>
.95, .31, 0, .81 ,.58,0, .60,.80,0, .36,.93,0, 0.,1.,0, -.36,.93,0, -.60,.80,0, -.81,.58,0, -.95,.31,0, -1.,0.,0, -.95,-
.31,0, -.81,-.58,0, -.60,-.80,0, -.36,-.93,0, 0.,-1.,0, .36,-.93,0, .60,-.80,0, .81,-.58,0, .95,-.31,0, 1.,0.,0.
                            </gml:coordinates>
                        </gml:LinearRing>
                    </gml:exterior>
                </gml:PolygonPatch>
                </gml:patches>
                </gml:Surface>
            </gml:surfaceMember>
            <gml:CompositeSurface>
        </gml:exterior>
</gml:Solid>
```

3.1.3 Spherical Interpolation

Geometry types that are not provided in GML can be created in an application schema. One rule that must followed is that the new geometry type must derive, directly or indirectly, from AbstractGeometryType. It is recommended that the new geometry type does not form the top-level gml:AbstractGeometryType, if possible, since this provides very limited information to GML-aware software. Instead try to choose the most specific geometry type possible in constructing your derived geometry types. For example, if the new geometry is some type of surface, then it should derive from gml:AbstractSurfaceType, or in the case that the surface uses a single uniform interpolation method, it may be more appropriate to derive from gml:AbstractSurfacePatchType.
A gml:SurfacePatch is of gml:AbstractSurfacePatchType and is just a higher dimensional analogue of gml:CurveSegment, i.e. it has a uniform interpolation type and always occurs as a patch of a gml:Surface. Figure 7 and the following schema fragment provide an example of a new type of surface patch called SphericalCap, which is just one of the two pieces of a sliced sphere.

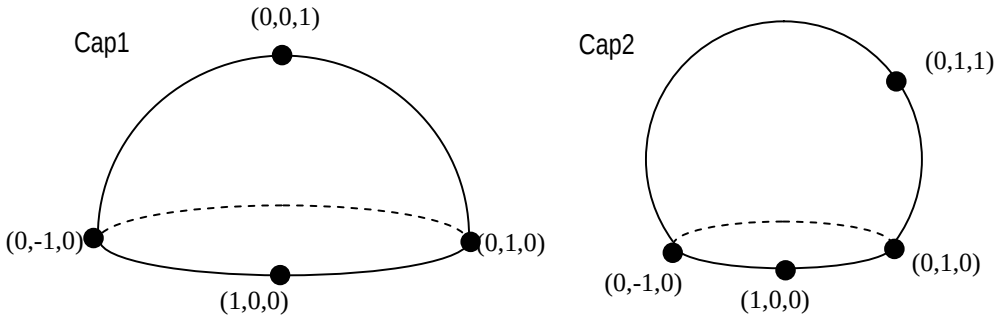

Figure 7. Two examples of a SphericalCap

```
<element name="SphericalCap" type="app:SphericalCapType" substitutionGroup="gml:_SurfacePatch"/>
<!-- ========================================================== -->
<complexType name="SphericalCapType">
    <complexContent>
        <extension base="gml:AbstractSurfacePatchType">
            <sequence>
                <choice>
                    <annotation>
                        <documentation>
                            The number of tuples in the coordinate list must be four.
```

```
                    </documentation>
                </annotation>
                <choice minOccurs="4" maxOccurs="4">
                    <element ref="gml:pos"/>
                    <element ref="gml:pointRep"/>
                </choice>
                <element ref="gml:coordinates"/>
            </choice>
        </sequence>
        <attribute name="interpolation" type="gml:SurfaceInterpolationType"
fixed="spherical"/>
      </extension>
    </complexContent>
  </complexType>
```

The SphericalCap is a higher dimensional analogue of Arc. The attribute "interpolation" specifies the interpolation mechanism used for this surface patch. The value of this attribute is fixed to "spherical"; i.e. the interpolation method will return a subset of points on a sphere that lie on one side of a circle that forms the boundary points of the spherical cap. The first 3 points are the control points of the boundary circle of the spherical cap, where circular interpolation is used. The fourth point is the last control point needed to uniquely determine the spherical cap. Spherical interpolation through the four points is used to complete the spherical cap. Note that the first 3 points cannot lie on a common line (not collinear), and the 4 points together cannot lie on a common plane (not coplanar). Sample instances of the SphericalCaps Cap1 and Cap2 in Figure 7 might be as follows:

```
<gml:Surface gml:id="Cap1">
    <gml:patches>
    <gml:SphericalCap>
        <gml:coordinates>0,-1,0 1,0,0 0,1,0 0,0,1</gml:coordinates>
    </gml:SphericalCap>
    </gml:patches>
</gml:Surface>

<gml:Surface gml:id="Cap2">
    <gml:patches>
    <gml:SphericalCap>
        <gml:coordinates>0,-1,0 1,0,0 0,1,0 0,1,1</gml:coordinates>
    </gml:SphericalCap>
    </gml:patches>
</gml:Surface>
```

3.2 Topology

Topology is a branch of mathematics that describes the properties of spatial objects that remain unchanged after “twisting” and “stretching” (continuous deformation). In GML, spatial topology is modeled using basic building blocks—nodes, edges, faces, and solids—called topology primitives, together with a description of their connective relationships to one another. The GML topology primitive types, Node, Edge, Face, and TopoSolid, are often used to represent the geometry primitives, Point, Curve, Surface, and Solid, respectively. The connectedness of nodes, coincidence of edges, and adjacency of faces and solids are some of the connective relationships between primitives that topology is concerned with. Unlike GML geometry, topology does not encode any coordinate values—so <pos> and <coordinates> tags are absent. The topology model is not concerned with the position of nodes, direction of edges, nor the shape of faces and solids.

3.2.1 Nodes, Edges and Faces

Figure 8 shows an example of a simple topology model for a road network. This example has three Nodes (A, B, C), four Edges (a, b, c, d), and two Faces (F1, F2).

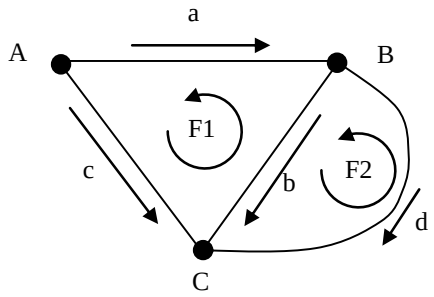

Figure 8. A Simple Two-dimensional Topology Network

The simplest encoding of a GML Node is as follows:

```
<gml:Node gml:id="A"/>
<gml:Node gml:id="B"/>
<gml:Node gml:id="C"/>
```

The Nodes above are uniquely identified using the gml:id attribute. The Edges can be encoded as follows:

```
<gml:Edge gml:id="a">
    <gml:directedNode orientation="-" xlink:href="#A"/>
    <gml:directedNode orientation="+" xlink:href="#B"/>
</gml:Edge>
<gml:Edge gml:id="b">
    <gml:directedNode orientation="-" xlink:href="#B"/>
    <gml:directedNode orientation="+" xlink:href="#C"/>
</gml:Edge>
<gml:Edge gml:id="c">
    <gml:directedNode orientation="-" xlink:href="#A"/>
    <gml:directedNode orientation="+" xlink:href="#C"/>
</gml:Edge>
<gml:Edge gml:id="d">
    <gml:directedNode orientation="-" xlink:href="#B"/>
    <gml:directedNode orientation="+" xlink:href="#C"/>
</gml:Edge>
```

In this example, the xlink:href attribute is used to reference Nodes that were previously defined. The orientation attribute is used to assign a negative orientation "-" to some of the nodes signifying that these nodes are at the start of the corresponding edge and positive orientation "+" to signify that these nodes are at the end of the edge. Note that each Edge is directed by its start and end node and can be traversed positively or negatively; the directed edge "+a" corresponds to traversing the path along "a" from A to B, and "–a" traverses the path from B to A. The directed edge "-a" can be encoded in GML using the directedEdge property with an orientation attribute "-", as shown below:

```
<gml:directedEdge orientation="-" xlink:href="#a"/>
```

Any route from A to B can be expressed as a TopoCurve in GML, which contains a list of directed edges {+c, -b} forming a connected path. The following example shows how to encode this path from A to B in GML as a TopoCurve:

```
<gml:TopoCurve>
    <gml:directedEdge orientation="+" xlink:href="#c"/>
    <gml:directedEdge orientation="-" xlink:href="#b"/>
</gml:TopoCurve>
```

In the GML topology model, each face is defined by its boundary, which consists of a list of directed edges. The directed edges in the boundary of each face are traversed in a counter-clockwise direction (following the convention in ISO TC 211/IS 19107 *Spatial Schema*) as indicated by the arrow around F1 and F2 in Figure 8. The orientation of each directed edge in the boundary of a face is either "+" or "-", depending on whether the inherent direction of the edge agrees or disagrees with the counter-clockwise orientation of the face. For example, the boundary of the Face labelled F1, when traversed counter-clockwise, consists of the directed edges in the set {c,-b,-a}. The faces are encoded in GML as follows:

```
<gml:Face gml:id="F1">
    <gml:directedEdge orientation="+" xlink:href="#c"/>
    <gml:directedEdge orientation="-" xlink:href="#b"/>
    <gml:directedEdge orientation="-" xlink:href="#a"/>
</gml:Face>
<gml:Face gml:id="F2">
    <gml:directedEdge orientation="+" xlink:href="#b"/>
    <gml:directedEdge orientation="-" xlink:href="#d"/>
</gml:Face>
```

In Figure 8, b is the only Edge that has a face on either side of it; that is, both faces F1 and F2 contain the directedEdge b—with either a positive or negative orientation—in their boundary lists. In this case, the faces F1 and F2 are said to be in the coboundary of b. In GML 3, the Edge primitive has an optional property called directedFace, whose value is a Face that is in the co-boundary of the Edge. To distinguish between the face on the left of b and the face on the right of b, each of the coboundary faces is assigned an orientation. A positive orientation corresponds to the left face and a negative orientation corresponds to the right face. Note that if the orientation of a directedFace in the coboundary of an Edge is "+", then the Face must contain the directedEdge with the same orientation "+" in its boundary list of directed edges. The encoding of the Edge b that describes its coboundary information in addition to its boundary information is as follows:

```
<gml:Edge gml:id="b">
    <gml:directedNode orientation="-" xlink:href="#B"/>
    <gml:directedNode orientation="+" xlink:href="#C"/>
    <gml:directedFace orientation="-" xlink:href="#F1"/>
    <gml:directedFace orientation="+" xlink:href="#F2"/>
</gml:Edge>
```

Similarly, each Node can be encoded with a co-boundary list of directedEdges to represent the edges that are incident upon the Node. A positive orientation on directedEdge corresponds to an edge that points towards the Node, and a negative orientation corresponds to an edge emanating from the Node. For example, the Node B from Figure 8 can be encoded as:

```
<gml:Node gml:id="B">
    <gml:directedEdge orientation="+" xlink:href="#a"/>
    <gml:directedEdge orientation="-" xlink:href="#b"/>
    <gml:directedEdge orientation="-" xlink:href="#d"/>
</gml:Node>
```

3.2.2 More Complex Topology Primitives

The coboundary of a Face is a list of directed TopoSolids. For example, consider the Face F that represents the equatorial plane of the solid ball in Figure 9.

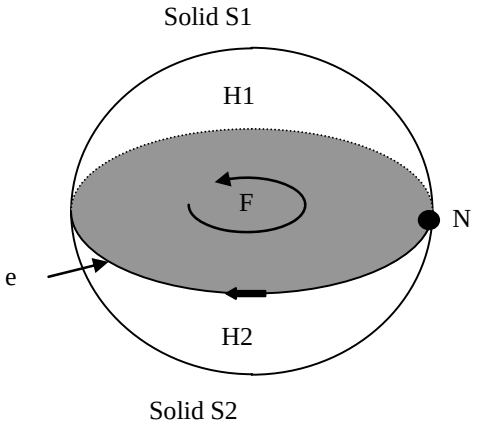

Figure 9. Two Solid half-balls in the Co-Boundary of a Face

The solid upper half-ball S1 and the solid lower half-ball S2 together form the solid ball as shown in Figure 9. The boundary of S1 consists of two faces, representing the upper hemisphere H1 and the equatorial plane F. The Solid S2 also has two faces in its boundary, F (with opposite orientation as used with S1) and the lower hemisphere H2. The Face F and the two TopoSolids S1 and S2 can be encoded as follows:

```
<gml:Face gml:id="F">
    <gml:directedEdge orientation="-">
        <gml:Edge gml:id="e">
            <gml:directedNode orientation="-">
                <gml:Node gml:id="N"/>
            </gml:directedNode>
            <gml:directedNode orientation="+" xlink:href="#N"/>
        </gml:Edge>
    </gml:directedEdge>
    <gml:directedTopoSolid orientation="+" xlink:href="#S1"/>
    <gml:directedTopoSolid orientation="-" xlink:href="#S2"/>
</gml:Face>

<gml:TopoSolid gml:id=”S1”>
    <gml:directedFace orientation="+">
        <gml:Face gml:id="H1">
            <gml:directedEdge orientation="+" xlink:href="#e"/>
        </gml:Face>
    </gml:directedFace>
    <gml:directedFace orientation="+" xlink:href="#F"/>
</gml:TopoSolid>

<gml:TopoSolid gml:id="S2">
    <gml:directedFace orientation="+">
        <gml:Face gml:id="H2">
            <gml:directedEdge orientation="+" xlink:href="#e"/>
        </gml:Face>
    </gml:directedFace>
    <gml:directedFace orientation="-" xlink:href="#F"/>
</gml:TopoSolid>
```

The shaded face in Figure 10 shows a more complicated use of two-dimensional topology. The face F3 has an exterior boundary Edge e1 and interior boundary edges e2, e3, e4, and e5. There is also an isolated Node N5 in the interior of F3 that is encoded using the gml:isolated property.

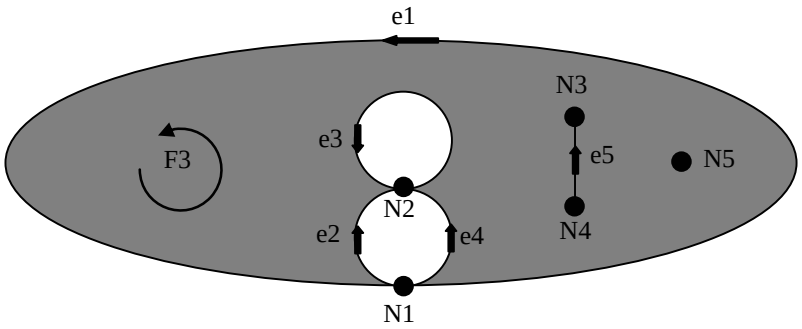

Figure 10. A Complex Face Configuration

The directed edges "+e1" and "-e3" each form a boundary ring (a simple closed loop in the boundary) of the face F3 that agrees with the counter-clockwise orientation of F3. The directed edges "+e2" and "-e4" together form another boundary ring of F3 in agreement with the orientation of F3, and likewise, the directed edges "+e5" and "-e5" together form another boundary ring. Notice that the "dangling" edge e5 has F3 on both the left and the right, and thus e5 occurs twice as a directedEdge in the following definition of F3, once with positive orientation and once with negative orientation. The boundary ring {+e1} in this case is referred to as the "exterior" boundary ring of F3, and the boundary rings formed by the sets {-e3}, {+e2, -e4}, {+e5, -e5} are all considered "interior" boundary rings following the convention used by ISO TC 211/DIS 19107 *Spatial Schema*. Note that the notions of "interior" and "exterior" in the context of boundaries are not explicitly encoded in GML Topology, but they are in GML Geometry. Note also that any number of boundary rings can intersect at a common Node, but boundary rings are not allowed to intersect along a common Edge. The Face F3 is encoded in GML as follows:

```
<gml:Face gml:id="F3">
    <gml:isolated>
        <gml:Node gml:id="N5"/>
    </gml:isolated>
    <gml:directedEdge orientation="+">
        <gml:Edge gml:id="e1">
            <gml:directedNode orientation="-">
                <gml:Node gml:id="N1"/>
            </gml:directedNode>
            <gml:directedNode orientation="+" xlink:href="#N1"/>
        </gml:Edge>
    </gml:directedEdge>
    <gml:directedEdge orientation="+">
        <gml:Edge gml:id="e2">
            <gml:directedNode orientation="-">
                <gml:Node gml:id="N1"/>
            </gml:directedNode>
            <gml:directedNode orientation="+" xlink:href="#N2"/>
        </gml:Edge>
    </gml:directedEdge>
    <gml:directedEdge orientation="-">
        <gml:Edge gml:id="e3">
            <gml:directedNode orientation="-" xlink:href="#N2"/>
            <gml:directedNode orientation="+" xlink:href="#N2"/>
        </gml:Edge>
    </gml:directedEdge>
    <gml:directedEdge orientation="-">
        <gml:Edge gml:id="e4">
```

```
                <gml:directedNode orientation="-">
                    <gml:Node gml:id="N1"/>
                </gml:directedNode>
                <gml:directedNode orientation="+">
                    <gml:Node gml:id="N2"/>
                </gml:directedNode>
            </gml:Edge>
        </gml:directedEdge>
        <gml:directedEdge orientation="-">
            <gml:Edge gml:id="e5">
                <gml:directedNode orientation="-">
                    <gml:Node gml:id="N4"/>
                </gml:directedNode>
                <gml:directedNode orientation="+">
                    <gml:Node gml:id="N3"/>
                </gml:directedNode>
            </gml:Edge>
        </gml:directedEdge>
        <gml:directedEdge orientation="+" xlink:href="#e5"/>
    </gml:Face>
```

3.2.3 Geometric Realizations of Topology

In GML, there is symmetry between the geometric and topological primitives. The geometry primitives—Point, Curve, Surface, and Solid—are often referred to as geometric realizations of—Node, Edge, Face, and TopoSolid, respectively. These geometric realizations are expressed using the following properties: pointProperty, curveProperty, surfaceProperty and solidProperty. For example, suppose coordinates are assigned to the following simple topology network in Figure 11 as shown.

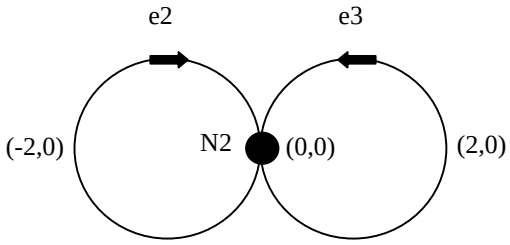

Figure 11. A geometric realization of topology.

The simple topology network can be encoded including with a geometric realization as follows:

```
    <gml:Node gml:id="N2">
        <gml:pointProperty>
            <gml:Point>
                <gml:coordinates>0,0</gml:coordinates>
            </gml:Point>
        </gml:pointProperty>
    </gml:Node>

    <gml:Edge gml:id="e2">
        <gml:directedNode orientation="-" xlink:href="#N2"/>
        <gml:directedNode orientation="+" xlink:href="#N2"/>
        <gml:curveProperty>
            <gml:Curve>
```

```
                <gml:segments>
                    <gml:Circle>
                        <gml:coordinates>0,0 –2,0 -1,1</gml:coordinates>
                    </gml:Circle>
                </gml:segments>
            </gml:Curve>
        </gml:curveProperty>
</gml:Edge>

<gml:Edge gml:id="e3">
        <gml:directedNode orientation="-" xlink:href="#N2"/>
        <gml:directedNode orientation="+" xlink:href="#N2"/>
        <gml:curveProperty>
            <gml:Curve>
                <gml:segments>
                    <gml:Circle>
                        <gml:coordinates>0,0 2,0 1,1</gml:coordinates>
                    </gml:Circle>
                </gml:segments>
            </gml:Curve>
        </gml:curveProperty>
</gml:Edge>
```

3.3 Coverages

A GML coverage is a distribution function, defined on a domain, which usually consists of a set of geometry elements. For example, the geometry elements may be a set of polygons in a surface tessellation, a set of curves segments along curve, a discrete set of points, or a rectified grid. The range of the distribution function could take on values such as elevation, temperature, pressure, rock type, or reflectance as shown in Figure 12.

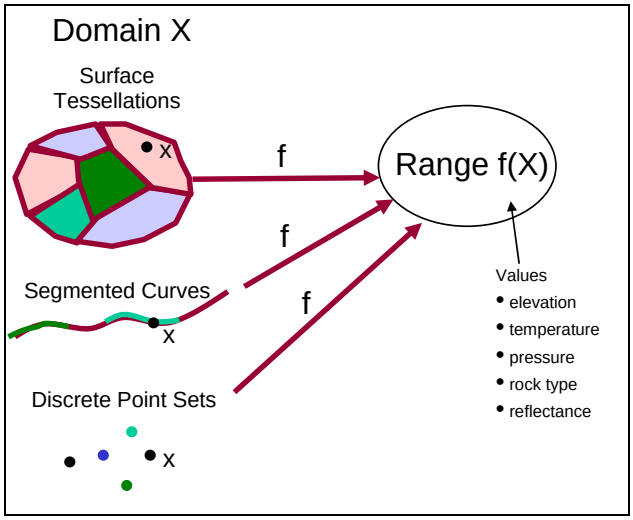

Figure 12. Coverages are distribution functions defined on some domain.

There are three components that define a GML coverage—the domain, range, and coverage function. These components are contained by the three properties: domainSet, rangeSet and coverageFunction, respectively. The GML spatial coverage model is represented by the Entity Relationship (ER) diagram in Figure 13.

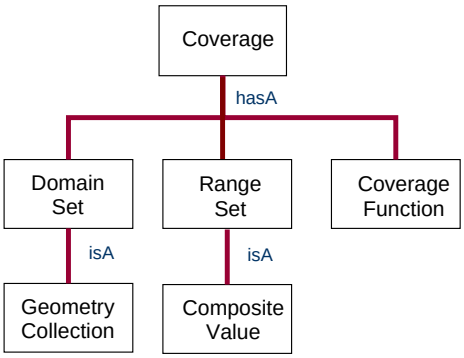

Figure 13. Entity Relationship (ER) View of a Coverage

3.3.1 Domain Set

In a GML instance, the value of the domainSet property will often be a GML geometry aggregate such as MultiPoint, MultiCurve, MultiPolygon, or a Grid such as RectifiedGrid.

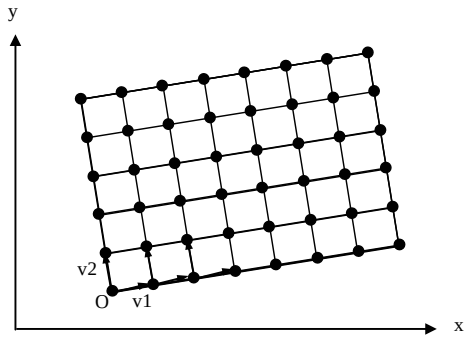

Figure 14. A Two-Dimensional Rectified Grid with Offset Vectors v1 and v2

In Figure 14, the origin is denoted as O, and the offset vectors are v1 and v2. The origin and offset vectors must all be of the same dimension—usually 2D or 3D. The GML RectifiedGrid has the properties: limits, axisName, origin and offsetVector. The value of limits is a GridEnvelope, which has two properties called low and high. The value of low is an integer list, for example [n1,n2], that represents the "lower left" corner O+n1v1+n2v2 of the grid. Similarly, high represents the "upper right" corner of the grid. The following listing shows two sample encodings of a domainSet containing a GML RectifiedGrid and a MultiPoint, respectively:

```
<gml:domainSet>
    <gml:RectifiedGrid dimension="2">
        <gml:limits>
    <gml:GridEnvelope>
            <gml:low>0 0</gml:low>
            <gml:high>9 6</gml:high>
            </gml:GridEnvelope>
        </gml:limits>
        <gml:axisName>x</gml:axisName>
        <gml:axisName>y</gml:axisName>
        <gml:origin>
            <gml:Point gml:id="O" srsName="urn:epsg:v6.1:coordinateReferenceSystem:1234">
                <gml:pos>2,1</gml:pos>
```

```
                </gml:Point>
            </gml:origin>
            <gml:offsetVector srsName="urn:epsg:v6.1:coordinateReferenceSystem:1234">1,
0.3</gml:offsetVector>
            <gml:offsetVector srsName="urn:epsg:v6.1:coordinateReferenceSystem:1234">-0.3,
1</gml:offsetVector>
        </gml:RectifiedGrid>
</gml:domainSet>

<gml:domainSet>
        <gml:MultiPoint srsName="urn:epsg:v6.1:coordinateReferenceSystem:1234">
            <gml:pointMembers>
        <gml:Point>
                <gml:pos>1 0</gml:pos>
                </gml:Point>
        <gml:Point>
                <gml:pos>0 1</gml:pos>
                </gml:Point>
        <gml:Point>
                <gml:pos>3 1</gml:pos>
                </gml:Point>
        <gml:Point>
                <gml:pos>1 2</gml:pos>
                </gml:Point>
        <gml:Point>
                <gml:pos>3 4</gml:pos>
                </gml:Point>
        <gml:Point>
                <gml:pos>1 5</gml:pos>
                </gml:Point>
            </gml:pointMembers>
        </gml:MultiPoint>
</gml:domainSet>
```

3.3.2 Range Set

The range set can be encoded as an aggregate value, a data block, or a binary file. An aggregate value encoding is the most verbose; the binary encoding is the most efficient, and the data block encoding lies somewhere in between. This section provides examples of the three different rangeSet encodings with temperature and pressure values. Note that the temperature and pressure values in these examples are defined in a GML application schema.

3.3.2.1 Aggregate Value

Any of the value aggregates defined in GML can be used to encode range data. Some commonly used aggregate values defined in the GML core schema valueObjects.xsd are as follows:

1. A (set of) ValueArray(s) in which the members of each array are homogeneously typed values
2. A member of the gml:_ScalarValueList substitution group, such as CategoryList or CountList

The following example shows how temperature and pressure values can be encoded as a set of value arrays:

```
<gml:rangeSet>
        <gml:ValueArray>
            <gml:valueComponents>
                <app:Temp uom="urn:x-si:v1999:uom:degreesC">0</app:Temp>
```

```
                <app:Temp uom="urn:x-si:v1999:uom:degreesC">24</app:Temp>
                <app:Temp uom="urn:x-si:v1999:uom:degreesC">17</app:Temp>
                <app:Temp uom="urn:x-si:v1999:uom:degreesC">19</app:Temp>
                        ...
            </gml:valueComponents>
        </gml:ValueArray>
        <gml:ValueArray>
            <gml:valueComponents>
                <app:Pressure uom="urn:x-si:v1999:uom:kPa">101.1</app:Pressure>
                <app:Pressure uom="urn:x-si:v1999:uom:kPa">101.2</app:Pressure>
                <app:Pressure uom="urn:x-si:v1999:uom:kPa">101.3</app:Pressure>
                <app:Pressure uom="urn:x-si:v1999:uom:kPa">101.4</app:Pressure>
                        ...
            </gml:valueComponents>
        </gml:ValueArray>
    </gml:rangeSet>
```

3.3.2.2 Data Block

A gml:DataBlock consists of two properties, rangeParameters and tupleList, whose values describe the range of a coverage. The value of rangeParameters is of gml:_Value type from valueObjects.xsd, which describes the quantities in the tupleList, including the units of measure used.
The following example shows how temperature and pressure values can be encoded in a DataBlock:

```
<gml:rangeSet>
    <gml:DataBlock>
        <gml:rangeParameters>
            <gml:CompositeValue>
                <gml:valueComponents>
                    <app:Temp uom="urn:x-si:v1999:uom:degreesC">template</Temp>
                    <app:Pressure uom="urn:x-si:v1999:uom:kPa">template</Pressure>
                </gml:valueComponents>
            </gml:CompositeValue>
        </gml:rangeParameters>
    <gml:tupleList>0,101.1 24,101.2 17,101.3 19,101.4  ...</gml:tupleList>
    </gml:DataBlock>
</gml:rangeSet>
```

Note that the aggregate value object CompositeValue is the target of rangeParameters in the instance above, which has a valueComponents property. Temp and Pressure are value objects derived from gml:MeasureType that are encapsulated by the valueComponents property tags. The tupleList property contains a list of coordinate tuples, where the entries of the coordinate tuples provide the quantities of the range parameters.

3.3.2.3 Binary File

The most efficient encoding of quantities in the range set is the binary file encoding provided by gml:File. The following example shows how to use gml:File to record the temperature and pressure values:

```
<gml:rangeSet>
    <gml:File>
        <gml:rangeParameters>
            <gml:CompositeValue>
                <gml:valueComponents>
```

```
                        <app:Temp uom="urn:x-si:v1999:uom:degreesC">template</app:Temp>
                        <app:Pressure uom="urn:x-si:v1999:uom:kPa">template</app:Pressure>
                    </gml:valueComponents>
                </gml:CompositeValue>
            </gml:rangeParameters>
            <gml:fileName>weather.dat</gml:fileName>
            <gml:fileStructure>Record Interleaved</gml:fileStructure>
        </gml:File>
    </gml:rangeSet>
```

The rangeParameters property is the same as that used for the DataBlock example. The properties fileName and fileStructure contain the location, name, and structure of the binary file in which the data is stored. Note that the fields within each record of the file correspond to the valueComponents of the CompositeValue in the given order.

Another sample instance of rangeSet that makes use of different range parameters is as follows:

```
<gml:rangeSet>
    <gml:File>
        <gml:rangeParameters>
            <gml:CompositeValue>
                <gml:valueComponents>
                        <app:SoilType codeSpace="urn:x-
NID:v0.1:soils:123">template</app:SoilType>
                        <app:SoilMoisture uom="urn:x-
IHSDM:v2.05a:uom:percent">template</app:SoilMoisture>
                </gml:valueComponents>
            </gml:CompositeValue>
        </gml:rangeParameters>
        <gml:fileName>soil.dat</gml:fileName>
        <gml:fileStructure>Record Interleaved</gml:fileStructure>
    </gml:File>
</gml:rangeSet>
```

3.3.3 Coverage Function

The value of gml:coverageFunction is a choice between gml:MappingRule or gml:GridFunction. The MappingRule is of type gml:StringOrRefType, which is a string or a URI reference to a mapping rule defined elsewhere. The mapping rule by default is linear if not specified, where linear mapping assigns the first geometry element (in document order) in the domain to the first value in the range, and so on.

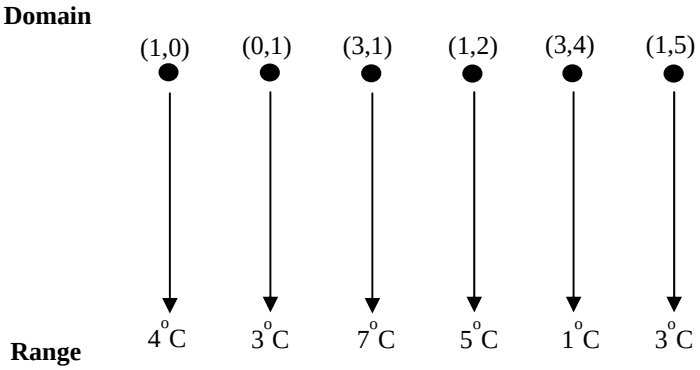

Figure 15. Linear Mapping Rule for a MultiPoint Temperature Coverage

3.3.3.1 Grid Functions

The GridFunction is used instead of MappingRule for grid coverages and has two properties, sequenceRule and StartPoint. The value of startPoint is an integer list that represents the first grid post in the grid to be traversed; the default start value is the value of low in GridEnvelope. The value of sequenceRule is one of the strings in the enumerated list sequence rules: Linear, Boustrophedonic, Cantor-diagonal, Spiral, Morton, or Hilbert. The sequenceRule property also has an optional order attribute whose value specifies one of the increment orders in the enumerated list: "+x+y", "+y+x", "x-y", "-x-y", in the case where the grid is 2-dimensional. The increment order of "+x+y" means that the grid points are traversed from lower to higher on the x-axis and from lower to higher on the y-axis. For example, Figure 16 and Figure 17 both illustrate the linear sequence rule but with different increment orders.

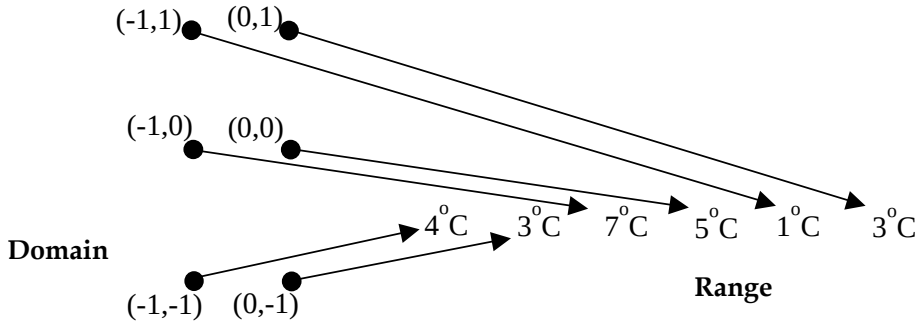

Figure 16. Linear Grid Function for a Grid Coverage with "+x+y" Increment Order

Note that in Figure 16, the starting point (-1,-1) is the same as the value of low in the corresponding GridEnvelope. The increment order "+x-y" means the grid points are traversed from lower to higher on the x-axis and from higher to lower on the y-axis as illustrated in Figure 17.

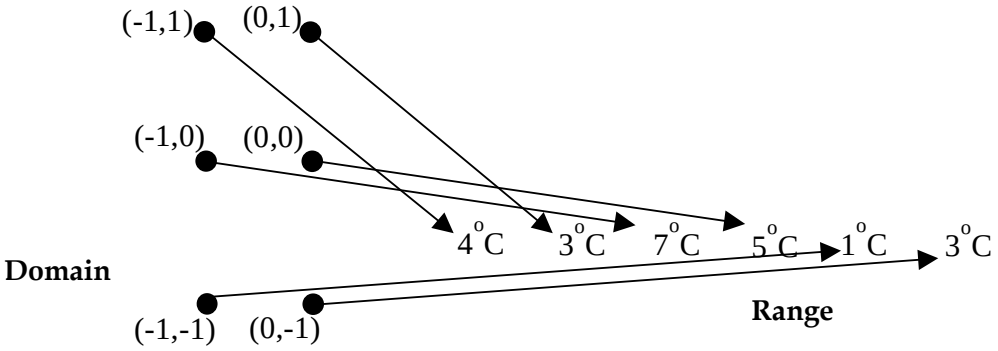

Figure 17. Linear Grid Function for a Grid Coverage with "+x-y" Increment Order

As shown in Figure 17, the starting point (-1,1) is the not the same as the value of low in the corresponding GridEnvelope. In this case the startPoint property is required as shown in the following instance:

```
<gml:domainSet>
    <gml:Grid dimension="2">
        <gml:limits>
            <gml:GridEnvelope>
                <gml:low>-1 -1</gml:low>
```

```
                    <gml:high>0 1</gml:high>
                </gml:GridEnvelope>
            </gml:limits>
            <gml:axisName>x</gml:axisName>
            <gml:axisName>y</gml:axisName>
        </gml:Grid>
</gml:domainSet>
<rangeSet>...</rangeSet>
<coverageFunction>
        <GridFunction>
            <sequenceRule order="+x-y">Linear</sequenceRule>
            <startPoint>0 1</startPoint>
        </GridFunction>
<coverageFunction>
```

Note that if the startPoint was not specified, the default starting point would have been the value of low, (-1,-1).

3.3.3.2 Encoding a Rectified Grid

The following sample instance shows the entire temperature and pressure coverage, TempPressure, with range data encoded as a DataBlock:

```
<app:TempPressure>
    <gml:rectifiedGridDomain>
        <gml:RectifiedGrid dimension="2">
            <gml:limits>
            <gml:GridEnvelope>
                <gml:low>-1 -1</gml:low>
                <gml:high>2 2</gml:high>
            </gml:GridEnvelope>
            </gml:limits>
            <gml:axisName>u</gml:axisName>
            <gml:axisName>v</gml:axisName>
            <gml:origin>
            <gml:Point gml:id="O" srsName="urn:epsg:v6.1:coordinateReferenceSystem:1234">
                <gml:coordinates>25,27</gml:coordinates>
            </gml:Point>
            </gml:origin>
            <gml:offsetVector>1, 0.2</gml:offsetVector>
            <gml:offsetVector>-0.2, 1</gml:offsetVector>
        </gml:RectifiedGrid>
    </gml:rectifiedGridDomain>
    <gml:rangeSet>
        <gml:DataBlock>
            <gml:rangeParameters>
            <gml:CompositeValue>
                <gml:valueComponents>
                    <app:Temp uom="urn:x-si:v1999:uom:degreesC">template</app:Temp>
                    <app:Pressure uom="urn:x-si:v1999:uom:kPa">template</app:Pressure>
            </gml:valueComponent>
            </gml:CompositeValues>
        </gml:rangeParameters>
            <gml:tupleList>3,101.2 5,101.3 7,101.4 11,101.5 13,101.6 17,101.7 19,101.7 23,101.8
29,101.9 31,102.0 37,102.1 41,102.2 43,102.3 47,102.0 53,102.5 59,102.6</gml:tupleList>
        </gml:DataBlock>
```

```
        </gml:rangeSet>
    </app:TempPressure>
```

3.4 Coordinate Reference Systems

A Coordinate Reference System (CRS) describes the spatial context of coordinates. A CRS consists of a Coordinate System (CS) that is related to the real world by a datum. In practice, a coordinate reference system is often referenced from a CRS dictionary by a geometry element in a data instance. For that purpose, a GML attribute called srsName is built into the base geometry type and inherited by every concrete geometry type. The srsName attribute has anyURI type, and the value of the attribute is a URI that may point to an entry in a CRS dictionary. The following example shows a geometry instance that references a location independent CRS dictionary using a URN.

```
<MultiPoint gml:id="MP1" srsName="urn:epsg:v6.1:coordinateReferenceSystem:1234">
    <pointMembers>
        <Point gml:id="P1">
            <coordinates>23378428,25959999</coordinates>
        </Point>
        <Point gml:id="P2">
            <coordinates>43411784,25959743</coordinates>
        </Point>
    </pointMembers>
</MultiPoint>
```

If the srsName attribute is not provided on a geometry element (such as on the Point geometries above), then the CRS must be specified as part of the larger context this geometry is part of, for example on the aggregate MultiPoint.

CRS dictionary entries can be created in a GML instance document using the GML types defined in the CRS and supporting schemas. Some of the commonly used types provided in the CRS schema are GeographicCRSType, GeocentricCRSType, EngineeringCRSType and ProjectedCRSType.

3.5 Temporal Components and Dynamic Features

In GML, features with dynamic properties include forest-fires, flood boundaries, vehicles, and evolving disaster situations. This section summarizes the temporal components that play a role in describing the dynamic properties of features.

3.5.1 Temporal Primitives

The two geometric time primitives provided in GML are TimeInstant and TimePeriod. TimeInstant is used to represent positions in time, while TimePeriod represents temporal length or duration. An example instance demonstrating the use of TimeInstant is as follows:

```
<gml:TimeInstant gml:id="T1">
  <gml:timePosition>
       2003-01-01T12:34:01
  </gml:timePosition>
</gml:TimeInstant>
```

The timePosition value "2003-01-01T12:34:01" is an XML dateTime data type, which follows the format defined in the ISO 8601 reference system (Gregorian Calendar). The TimePeriod primitive is typically used to encode an interval of time between two TimeInstants. In GML instances, a TimePeriod must have either a begin or an end property. Each property has a child TimeInstant element that provides the time values for the beginning or end of the period. If only one of these properties is provided, then a duration must also be

included to implicitly specify the position of the other one. For example, an interval that begins at 2003-01-01 and has a duration of six days ends at 2003-01-07.

```
<gml:TimePeriod>
    <gml:begin>
        <gml:TimeInstant>
            <gml:timePosition>
                2002-01-01T21:29-08:00
        </gml:timePosition>
        </gml:TimeInstant>
    </gml:begin>
    <gml:end>
        <gml:TimeInstant>
            <gml:timePosition xlink:href="#T1"/>
        <gml:TimeInstant>
    </gml:end>
</gml:TimePeriod>
```

3.5.2 Temporal Reference Systems

Temporal reference systems, such as calendars, provide the context for time measurements. The default temporal reference system in GML is the same as the one defined in ISO 8601—the Gregorian calendar with Coordinated Universal Time (UTC). Other reference systems—such as the Global Positioning System (GPS) calendar and the Julian calendar—can also be used.

To reference a calendar, you need to specify a value in the frame attribute on one of the temporal primitives. The following example shows how to reference the Gregorian calendar from a gml:TimePosition property.

```
<gml:TimeInstant>
    <gml:timePosition frame="ISO-8601">...</gml:timePosition>
</gml:TimeInstant>
```

If the timePosition property does not specify a value for the frame attribute, then the property automatically references the Gregorian calendar, the default temporal reference system in GML. The frame attribute is also used to reference temporal ordinal reference systems and temporal coordinate reference systems.

3.6 Dynamic Features

Dynamic features can be defined in GML by deriving from DynamicFeatureType. Such a dynamic feature inherits a history property that is used to capture its evolution by recording a sequence of GML TimeSlices. Each TimeSlice object contains all of the properties of the feature that change over time. In the following sample application schema fragment the content model of a moving vehicle is defined.

```
<element name="Bus" type="app:BusType" substitutionGroup="gml:_Feature"/>

<complexType name="app:BusType">
  <complexContent>
    <extension base="gml:DynamicFeatureType">
        <sequence>
            <element name="maxPassengers" type="positiveInteger"/>
        </sequence>
   </extension>
    </complexContent>
</complexType>
```

In addition to inheriting the history property from DynamicFeatureType, the BusType also specifies an additional static property maxPassengers. A sample Bus instance might be as follows:

```
<Bus gml:id="B999">
  <history>
    <TimeSlice gml:id="TS-0">
      <gml:validTime>
        <gml:TimeInstant>
          <gml:timePosition>2003-02-24T14:22-8:00</gml:timePosition>
        </gml:TimeInstant>
      </gml:validTime>
      <gml:location>
        <gml:Point srsName="urn:epsg:v6.1:coordinateReferenceSystem:1234">
          <gml:pos>0 100</pos>
        </gml:Point>
      </gml:location>
      <numPassengers>25</numPassengers>
    </TimeSlice>
    <TimeSlice gml:id="TS-1">
      <gml:validTime>
        <gml:TimeInstant>
          <gml:timePosition>2003-02-24T15:11-8:00</gml:timePosition>
        </gml:TimeInstant>
      </gml:validTime>
      <gml:location>
        <gml:Point srsName="urn:epsg:v6.1:coordinateReferenceSystem:1234">
          <gml:pos>0 0</pos>
        </gml:Point>
      </gml:location>
      <numPassengers>20</numPassengers>
    </TimeSlice>
    <TimeSlice gml:id="TS-2">
      <gml:validTime>
        <gml:TimeInstant>
          <gml:timePosition>2003-02-24T15:22-8:00</gml:timePosition>
        </gml:TimeInstant>
      </gml:validTime>
      <gml:location>
        <gml:Point srsName="urn:epsg:v6.1:coordinateReferenceSystem:1234">
          <gml:pos>100 0</pos>
        </gml:Point>
      </gml:location>
      <numPassengers>15</numPassengers>
    </TimeSlice>
  </history>
```

```
    <maxPassengers>60</maxPassengers>
</Bus>
```

3.7 Units of Measure

In many geo-spatial applications, it is important to be able to associate units of measure with quantities. In practice, units of measure are often referenced from a dictionary. For this purpose, a GML attribute called uom is provided. The uom attribute is of anyURI type, and the value of the attribute is a URI that points to an entry in a unit dictionary. The following example shows an instance that references well known unit.

```
<app:RailCar gml:id="RC1">
    <app:length uom="urn:x-si:v1999:uom:metre">10</app:length>
        ...
</app:RailCar>
```

GML provides a set of schema components for defining units of measure and units of measure dictionaries. Units of measure dictionaries are based on the dictionary model that is defined in dictionary.xsd.

3.7.1 Unit Dictionaries

Base units are units of measure that are independent; that is, they cannot be expressed as a combination of other units. Base units are the most commonly used units for fundamental quantities such as length, mass, and time. The seven base units as defined by the International System of Units (SI) are: metres, seconds, amperes, Kelvin, mole, and candela.

A sample GML dictionary entry for the base unit "metre" can be encoded as follows:

```
<gml:Dictionary gml:id="unitsDictionary">
    ...
<gml:dictionaryEntry>
    <gml:DefinitionCollection gml:id="SIBaseUnits">
        <gml:description>The Base Units from the SI units system.</gml:description>
        <gml:name>SI Base Units</gml:name>
        <gml:dictionaryEntry>
        <gml:BaseUnit gml:id="metre">
            <gml:description>...</gml:description>
            <gml:name codeSpace="http://www.bipm.fr/en/3_SI/base_units.html">metre</gml:name>
            <gml:name xml:lang="en/US">meter</gml:name>
            <gml:quantityType>length</gml:quantityType>
            <gml:catalogSymbol codeSpace="http://www.bipm.fr/
                en/3_SI/base_units.html">m</gml:catalogSymbol>
            <gml:unitsSystem xlink:href="http://www.bipm.fr/en/3_SI"/>
        </gml:BaseUnit>
        </gml:dictionaryEntry>
            ...
        </DefinitionCollection>
    </gml:dictionaryEntry>
</gml:Dictionary>
```

Other units such as derived and conventional units can also be defined in GML in a similar way.

4 GML SCHEMA MANAGEMENT

GML application schemas are anticipated to be increasingly deployed on the Internet as the standard framework for sharing geographic data. As more schemas become available, it will be necessary to provide

more sophisticated mechanisms for managing schemas. Schema registry web services can fill this need by providing:

1. Online access to the GML application schemas for the appropriate domain
2. Administrative functions for the schema administrator such as authorization and versioning
3. Classification and other means for helping a user locate a schema

Schema registries will likely become the source for all of the GML application schemas that are supported by different geospatial web services. Clients, providers, and other services use the schemas that are stored on the GML application schema registry. These schemas provide the clients and providers with the feature types for request and response messages. Note that the schema registry and service registry are both metadata registries that can be part of an OGC Catalogue. The Catalogue is a key component in a common service architecture that manages shared resources and facilitates the discovery of resources within an open and distributed system.

5 REFERENCES

Lake, R., Burggraf, D.S., et. al. (2005) *GML: Foundation for the Geo-Web*, London, UK: Wiley & Sons.

Lake, R., Cox, S., et. al. (2004) OpenGIS® *Geography Markup Language Implementation Specification*, OGC 03-105r1, Version 3.1.1.

Vretanos, P.A. (2002) OpenGIS® *Web Feature Service Implementation Specification*, OGC 01-065r1,Version 1.0.0.