Geography Markup Language (GML) 2.0

OpenGIS® Implementation Specification, 20 February 2001

OGC Document Number: 01-029

This version:
http://www.opengis.net/gml/01-029/GML2.html
(Available as: PDF, zip archive of XHTML)
Latest version:
http://www.opengis.net/gml/01-029/GML2.html
Previous versions:
http://www.opengis.net/gml/00-029/GML.html

Editors:
Simon Cox (CSIRO Exploration & Mining) <Simon.Cox@dem.csiro.au>
Adrian Cuthbert (SpotOn MOBILE) <adrian@spotonmobile.com>
Ron Lake (Galdos Systems, Inc.) <rlake@galdosinc.com>
Richard Martell (Galdos Systems, Inc.) <rmartell@galdosinc.com>
Contributors:
Simon Cox (CSIRO Exploration & Mining) <Simon.Cox@dem.csiro.au>
Adrian Cuthbert (SpotOn MOBILE) <adrian@spotonmobile.com>
Paul Daisey (U.S. Census Bureau) <pdaisey@geo.census.gov>
John Davidson (OGC IP2000 Team) <georef@erols.com>
Sandra Johnson (MapInfo Corporation) <sandra_johnson@mapinfo.com>
Edric Keighan (Cubewerx Inc.) <ekeighan@cubewerx.com>
Ron Lake (Galdos Systems Inc.) <rlake@galdosinc.com>
Marwa Mabrouk (ESRI Ltd.) <mmabrouk@esri.com>
Serge Margoulies (IONIC Software) <serge.margoulies@ionicsoft.com>
Richard Martell (Galdos Systems, Inc.) <rmartell@galdosinc.com>
Lou Reich (NASA/CSC) <louis.i.reich@gsfc.nasa.gov>
Barry O'Rourke (Compusult Ltd.) <barry@compusult.nf.ca>
Jayant Sharma (Oracle Corporation) <jsharma@us.oracle.com>

Panagiotis (Peter) Vretanos (CubeWerx Inc.) <pvretano@cubewerx.com>

Abstract

The Geography Markup Language (GML) is an XML encoding for the transport and storage of geographic information, including both the spatial and non-spatial properties of geographic features. This specification defines the XML Schema syntax, mechanisms, and conventions that

- Provide an open, vendor-neutral framework for the definition of geospatial application schemas and objects;
- Allow profiles that support proper subsets of GML framework descriptive capabilities;
- Support the description of geospatial application schemas for specialized domains and information communities;
- Enable the creation and maintenance of linked geographic application schemas and datasets;
- Support the storage and transport of application schemas and data sets;
- Increase the ability of organizations to share geographic application schemas and the information they describe.

Implementers may decide to store geographic application schemas and information in GML, or they may decide to convert from some other storage format on demand and use GML only for schema and data transport.

Document status

This document is an OpenGIS® Implementation Specification.

XML instances which are compliant to this specification shall validate against a conforming application schema. A conforming application schema shall import the Geometry Schema (geometry.xsd), the Feature Schema (feature.xsd), and the XLinks schema (xlinks.xsd) as base schemas; furthermore, it shall be developed using the rules for the development of application schemas specified in section 5 of this document.

Sections 1 and 2 of this document present the background information and modeling concepts that are needed to understand GML. Section 3 presents the GML conceptual model which is independent of encoding. Section 4 presents material which discusses the encoding of the

GML conceptual model using the XML Schema definition language (XSDL). This material is intended to demonstrate how to employ the normative GML geometry and feature schemas specified in Appendices A and B of this document. Section 5 of this document presents the rules for the development of conformant GML application schemas. Section 6 presents examples to illustrate techniques for constructing compliant GML application to model recurring geographic themes; these techniques are not normative but they do represent the collective experience of the editors of this document and are strongly recommended. Conforming profiles of this document shall be developed according the the rules specified in section 7. Appendix A presents the Geometry schema, Appendix B presents the Feature schema, and Appendix C presents the XLinks schema.

The OpenGIS Consortium (OGC) invites comments on this Implementation Specification--please submit them to gml.sig@opengis.org.

Table of Contents

1 Representing geographic features

2: GML overview Go

1.1 Introduction

This section introduces the key concepts required to understand how the Geography Markup Language (GML) models the world. It is based on the OGC Abstract Specification (available online: http://www.opengis.org/techno/specs.htm), which defines a geographic feature as "an abstraction of a real world phenomenon; it is a geographic feature if it is associated with a location relative to the Earth." Thus a digital representation of the real world can be thought of as a set of features. The state of a feature is defined by a set of properties, where each property can be thought of as a {name, type, value} triple. The number of properties a feature may have, together with their names and types, are determined by its type definition. Geographic features are those with properties that may be geometry-valued. A feature collection is a collection of features that can itself be regarded as a feature; as a consequence a feature collection has a feature type and thus may have distinct properties of its own, in addition to the features it contains.

This specification is concerned with what the OGC calls *simple features*: features whose geometric properties are restricted to 'simple' geometries for which coordinates are defined in two dimensions and the delineation of a curve is subject to linear interpolation. While this release of GML does permit coordinates to be specified in three dimensions, it currently provides no direct support for three-dimensional geometry constructs. The term 'simple features' was originally coined to describe the functionality defined in the set of OpenGIS® Implementation Specifications (available online: http://www.opengis.org/techno/specs.htm);

GML follows the geometry model defined in these specifications. For example, the traditional 0, 1 and 2-dimensional geometries defined in a two-dimensional spatial reference system (SRS) are represented by points, line strings and polygons. In addition, the geometry model for simple features also allows geometries that are collections of other geometries (either homogeneous multi-point, multi-line string and multi-polygon collections, or heterogeneous geometry collections). In all cases the 'parent' geometry element is responsible for indicating in which SRS the measurements have been made.

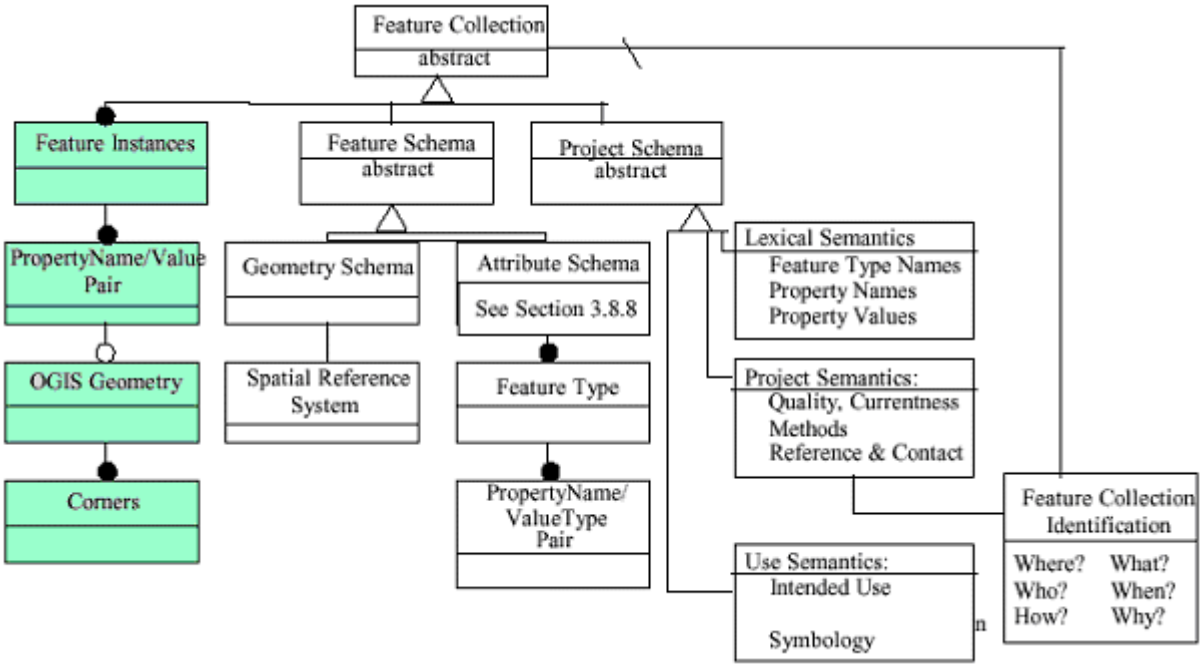
How can GML be used to represent real-world phenomena? Suppose somebody wishes to build a digital representation of the city of Cambridge in England. The city could be represented as a feature collection where the individual features represent such things as rivers, roads and colleges; such a classification of real world phenomena determines the feature types that need to be defined. The choice of classification is related to the task to which the digital representation will ultimately be put. The `River` feature type might have a property called *name* whose value must be of the type 'string'. It is common practice to refer to the typed property; thus the `River` feature type is said to have a string property called *name*. Similarly, the `Road` feature type might have a string property called *classification* and an integer property called *number*. Properties with simple types (e.g. integer, string, float, boolean) are collectively referred to as simple properties.

The features required to represent Cambridge might have geometry-valued properties as well as simple properties. Just like other properties, geometric properties must be named. So the `River` feature type might have a geometric property called *centerLineOf* and the `Road` feature type might have a geometric property called *linearGeometry*. It is possible to be more precise about the type of geometry that can be used as a property value. Thus in the previous examples the geometric property could be specialised to be a line string property. Just as it is common to have multiple simple properties defined on a single feature type, so too a feature type may have multiple geometric properties.

1.2 Feature and Geometry models

The abstract feature model used by the Open GIS Consortium is shown in Figure 1.1 using Syntropy notation. While it is common practice in the Geospatial Information (GI) community to refer to the properties of a feature as attributes, this document refers to them as properties in order to avoid potential confusion with the attributes of XML elements.

Figure 1.1: The abstract feature model



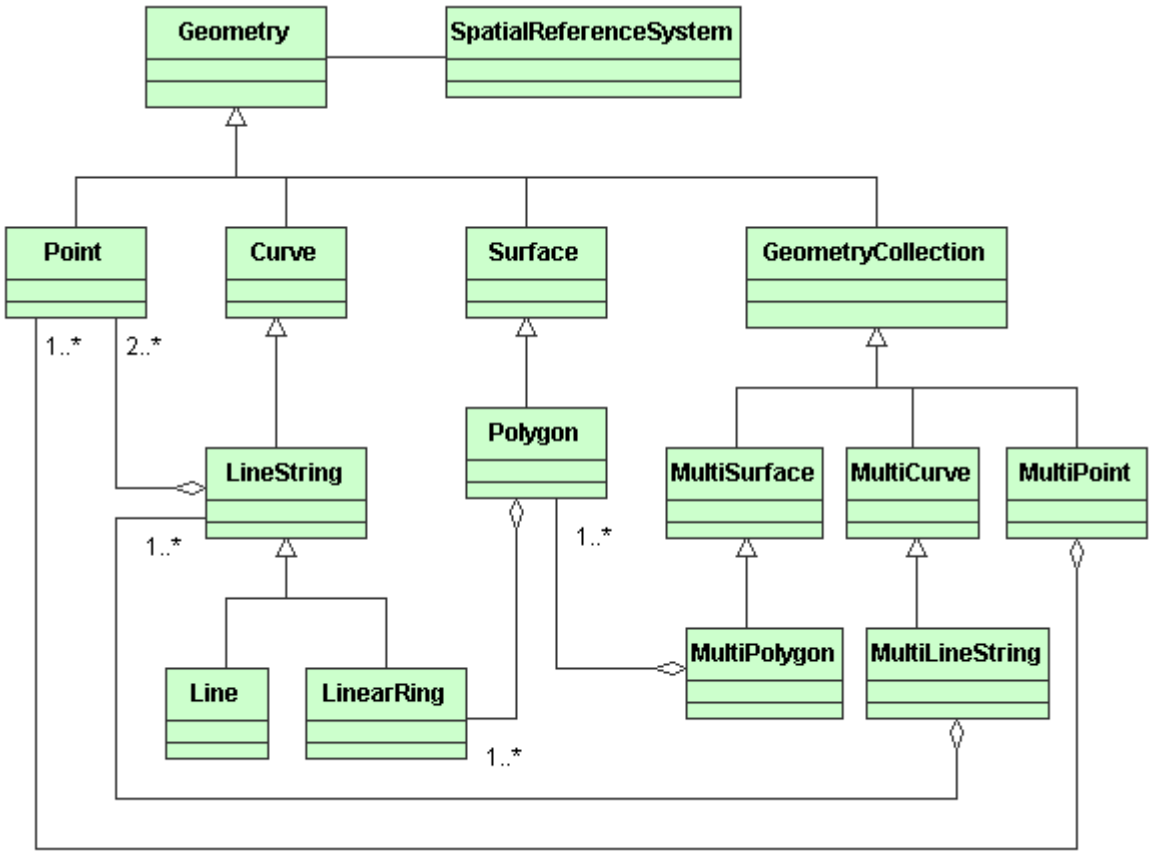
The 'Simple Features' model represents a simplification of the more general model described in the OpenGIS Abstract Specification, this simplification was the result of developing a number of implementation specifications. There are two major simplifications:

- Features are assumed to have either simple properties (booleans, integers, reals, strings) or geometric properties; and
- Geometries are assumed to be defined in two-dimensional SRS and use linear interpolation between coordinates.

A number of consequences follow from these simplifications. For example, simple features only provide support for vector data; and simple features are not sufficiently expressive to explicitly model topology. This version of GML addresses the first of these limitations in that it allows features to have complex or aggregate non-geometric properties. Such complex properties may themselves be composed of other complex and simple properties. Common examples include dates, times, and addresses. It is expected that future versions of GML will address the second of these limitations and provide more elaborate geometry models.

The geometry object model for simple features (Figure 1.2) has an (abstract) base Geometry class and associates each geometry object with an SRS that describes the coordinate space in which the object is defined. GML mirrors this class hierarchy but omits some intermediate (i.e. non-leaf) types such as Curve, Surface, MultiSurface, and MultiCurve.

Figure 1.2: The geometry model for simple features



2 Overview of GML

3: Conceptual framework Go

2.1 Design goals

GML was developed with a number of explicit design goals, a few of which overlap the objectives of XML itself:

- provide a means of encoding spatial information for both data transport and data storage, especially in a wide-area Internet context;
- be sufficiently extensible to support a wide variety of spatial tasks, from portrayal to analysis;
- establish the foundation for Internet GIS in an incremental and modular fashion;
- allow for the efficient encoding of geo-spatial geometry (e.g. data compression);
- provide easy-to-understand encodings of spatial information and spatial relationships, including those defined by the OGC Simple Features model;

- be able to separate spatial and non-spatial content from data presentation (graphic or otherwise);
- permit the easy integration of spatial and non-spatial data, especially for cases in which the non-spatial data is XML-encoded;
- be able to readily link spatial (geometric) elements to other spatial or non-spatial elements.
- provide a set common geographic modeling objects to enable interoperability of independently-developed applications.

GML is designed to support interoperability and does so through the provision of basic geometry tags (all systems that support GML use the same geometry tags), a common data model (features/properties), and a mechanism for creating and sharing application schemas. Most information communities will seek to enhance their interoperability by publishing their application schemas; interoperability may be further improved in some cases through the use of profiles as outlined in section 7.

2.2 Schemas for geospatial data

In general terms a schema defines the characteristics of a class of objects; in XML a schema also describes how data is marked up. GML strives to cater to a broad range of users, from neophytes to domain experts interested in modeling the semantics of geo-spatial information. Version 2.0 of GML is compliant with the XML Schema Candidate Recommendation published by the W3C in two parts on 24 October 2000 [XMLSchema1], [XMLSchema2]). GML has also been developed to be consistent with the XML Namespaces Recommendation [XMLName]. Namespaces are used to distinguish the definitions of features and properties defined in application-specific domains from one another, and from the core constructs defined in GML modules.

Geospatial feature types can be considered apart from their associated schemas. That is, a `Road` type (or class) exists independently of its schema definition whether it's expressed in terms of a DTD or an XML Schema. In GML 2.0 geospatial types are captured as element names, but these names assert the existence of model-level types separately from their XML Schema encodings. Consider the following example: a `Road` type is introduced in an application schema by declaring a global <Road> element of a type named `RoadType` (<element name="Road" type="RoadType"/>). We note the interplay of two perspectives: conceptual and implementation. Declaring that *width* is a property of a `Road` is a model-level assertion that says nothing about whether width is a floating point number with two decimal places or simply an integer--these are data and process characteristics at the implementation level.

GML 2.0 defines three base schemas for encoding spatial information. The Geometry schema

(geometry.xsd) replaces the DTD that appeared in GML 1.0. This release provides an enhanced Feature schema that supports feature collections (as feature types) and includes common feature properties such as *fid* (a feature identifier), *name* and *description*. The XLink schema provides the XLink attributes to support linking functionality. Database implementations are required to provide an application schema that defines the schema expressions for the geographic features that they support, and these are derived from the definitions found in the Feature schema.

The XML schema definition language provides a superset of the facilities provided by the DTD validation mechanism defined in the XML 1.0 specification. XML Schema provides a rich set of primitive datatypes (e.g. string, boolean, float, month), and it allows the creation of built-in and user-defined datatypes. XML Schema offers several advantages when it comes to constraining GML encodings:

- it enables the intermingling of different vocabularies using namespaces;
- it permits finer control over the structure of the type definition hierarchy; and
- it confers extensibility and flexibility via derived types and substitution groups

2.3 Graphical rendering

GML has been designed to uphold the principle of separating content from presentation. GML provides mechanisms for the encoding of geographic feature data without regard to how the data may be presented to a human reader. Since GML is an XML application, it can be readily styled into a variety of presentation formats including vector and raster graphics, text, sound and voice. Generation of graphical output such as maps is one of the most common presentations of GML and this can be accomplished in a variety of ways including direct rendering by graphical applets or styling into an XML graphics technology (e.g. SVG [SVG] or X3D [VRML200x]). It should be noted that GML is not dependent on any particular XML graphical specification.

3 The conceptual framework

4: Base schemas Go

3.1 Features and properties

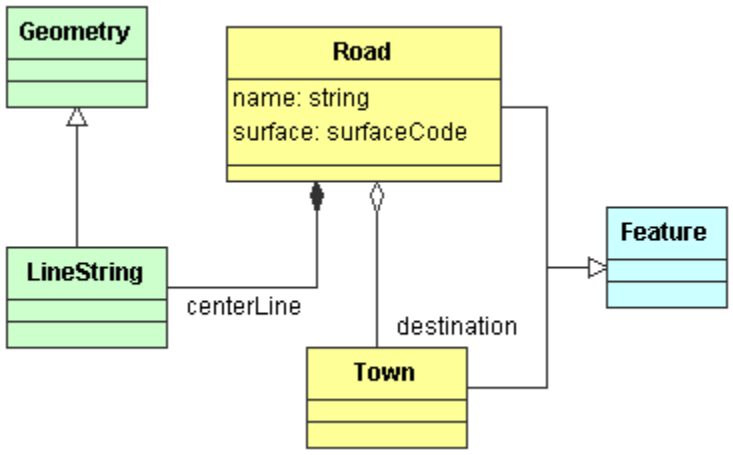
GML is an XML encoding for geographic features. In order to correctly interpret a GML document it is necessary to understand the conceptual model that underlies GML, which is described in the OGC Abstract Specification.

3.1.1 Object Model

A geographic feature is essentially a named list of properties. Some or all of these properties may be geospatial, describing the position and shape of the feature. Each feature has a *type*, which is equivalent to a *class* in object modeling terminology, such that the class-definition prescribes the named properties that a particular feature of that type is required to have. So a Road might be defined to have a name, a surface-construction, a destination, and a centreLine. The properties themselves are modeled in UML as *associations*, or as *attributes*, of the feature class. The feature property type is given by the *rolename* from an association, or by the *attribute name*. The *values* of the properties are themselves also instances of defined classes or types. So the Road name is a text-string, the surface-construction might be a text token selected from an enumerated list, the destination is another feature of type Town, and the centreLine is a LineString, which is a *geometry property*.

In UML it is partly a matter of taste whether a property is represented as an association or attribute, though it is common for a property with a complex or highly structured type to be modeled as an association, while simple properties are typically class attributes. If the value of a property only exists in the presence of the feature, such as the Road name, then it may use either a UML *composition association* or be represented as an attribute, as these two methods are functionally equivalent. However, if the value of a property is loosely bound to the object and the property value is an object that might exist independently of the feature, such as the Town that is the destination of a Road, then it must use a form of UML association called *aggregation* (see Figure 3.1).

Figure 3.1: Composition and aggregation relationships



3.1.2 XML encoding of the object model

A feature is encoded as an XML element whose name is the feature type according to some classification. The feature instance contains feature properties, each as an XML element whose name is the property name. Each of these contains another element whose name is the type of the property value or instance; this produces a "layered" syntax in which properties and instances are interleaved.

GML adopts a uniform coding convention to help distinguish properties from instances: element names that represent *instances* of GML classes start with an uppercase letter (e.g. Polygon), while tags that represent *properties* start with a lowercase letter; all embedded words in the property name start with uppercase letters (e.g. centerLineOf).

3.1.3 Functional view of the object model

From a functional perspective we can consider a property as a function that maps a feature onto a property value. A property is characterised by the input feature type and the type of the value that is returned. For example, suppose the feature type `House` has a String property called *address* and a Polygon property called *extentOf*. Using functional notation we can then write:

address(`House`) = String
extentOf(`House`)= Polygon

This approach can also be applied to feature collections that have features as members:

featureMember(`FeatureCollection`) = Feature

3.2 Geometric properties

In general the definition of feature properties lies in the domain of application schemas. However, since the OGC abstract specification defines a small set of basic geometries, GML defines a set of geometric property elements to associate these geometries with features.

The GML Feature schema also provides descriptive names for the geometry properties, encoded as common English language terms. Overall, there are three levels of naming geometry properties in GML:

1. **Formal names** that denote geometry properties in a manner based on the type of geometry allowed as a property value
2. **Descriptive names** that provide a set of standardised synonyms or aliases for the formal names; these allow use of a more user-friendly set of terms.
3. **Application-specific names** chosen by users and defined in application schemas based on GML

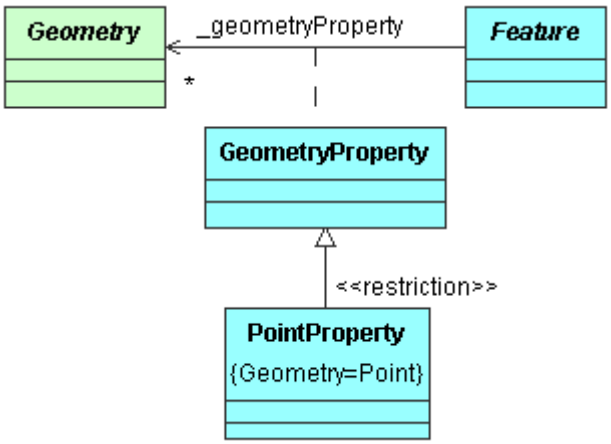
The formal and descriptive names for the basic geometric properties are listed in Table 3.1; these names appear in the Feature schema to designate common geometric properties. The precise semantics of these geometry properties (e.g. "What does position of an object mean?" or "Are location and position synonymous?") is not specified.

Table 3.1: Basic geometric properties

Formal name	Descriptive name	Geometry type
boundedBy	-	Box
pointProperty	location, position, centerOf	Point
lineStringProperty	centerLineOf, edgeOf	LineString
polygonProperty	extentOf, coverage	Polygon
geometryProperty	-	*any*
multiPointProperty	multiLocation, multiPosition, multiCenterOf	MultiPoint
multiLineStringProperty	multiCenterLineOf, multiEdgeOf	MultiLineString
multiPolygonProperty	multiExtentOf, multiCoverage	MultiPolygon
multiGeometryProperty	-	MultiGeometry

There are no inherent restrictions in the type of geometry property a feature type may have. For example, a `RadioTower` feature type could have a *location* that returns a Point geometry to identify its location, and have another geometry property called *extentOf* that returns a Polygon geometry describing its physical structure. A geometric property can be modeled in UML as an association class. Figure 3.2 illustrates how the geometryProperty relation associates an abstract feature type with an abstract geometry type.

Figure 3.2: Geometric properties as instances of an association class



In Figure 3.2 we also see that a *pointProperty* is a concrete instance of `GeometryProperty` that links a feature instance with a <Point> instance. An important point needs to be emphasized here: GML uses property elements to carry the role name of an association; this preserves important semantic relationships that would otherwise be difficult--or impossible--to infer. Such a practice also helps to maintain congruence between a GML schema and its corresponding UML model (if one exists).

3.3 Application schemas

Three base XML Schema documents are provided by GML: feature.xsd which defines the general feature-property model, geometry.xsd which includes the detailed geometry components, and xlinks.xsd which provides the XLink attributes used to implement linking functionality. These schema documents alone do *not* provide a schema suitable for constraining data instances; rather, they provide base types and structures which may be used by an *application schema*. An application schema declares the actual feature types and property types of interest for a particular domain, using components from GML in standard ways. Broadly, these involve defining application-specific types which are derived from types in the standard GML schemas, or by directly including elements and types from the standard GML schemas.

The base GML schemas effectively provide a meta-schema, or a set of foundation classes, from which an application schema can be constructed. User-written application schemas may declare elements and/or define types to name and distinguish significant features and feature collections from each other; the methods used to accomplish this are presented in section 4. A set of (normative) guidelines and rules for developing an application schema which conforms with GML are given in section 5. By following these rules and deriving the types defined in an application schema from components in the base GML schemas, the application schema benefits from standardised constructs and is guaranteed to conform to the OGC Feature

Model. A number of complete examples appear in section 6.

The GML Geometry schema is listed in Appendix A. The <import> element in the Geometry schema brings in the definitions and declarations contained in the XLinks schema. The GML Geometry schema includes type definitions for both abstract geometry elements, concrete (multi) point, line and polygon geometry elements, as well as complex type definitions for the underlying geometry types.

The GML Feature schema is listed in Appendix B. The <include> element in the Feature schema brings in the definitions and declarations contained in the Geometry schema; like the geometry schema, the Feature schema defines both abstract and concrete elements and types.

4 Encoding GML

5: Application schemas Go

4.1 Introduction

This section describes how to encode geospatial (geographic) features in GML. The encoding of spatial features using GML 2.0 requires the use of two XML Schemas: the GML Feature Schema (feature.xsd) and the GML Geometry Schema (geometry.xsd); with these two simple schemas it is possible to encode a wide variety of geospatial information.

The remainder of this sub-section introduces the two XML Schemas using UML notation. The following sub-sections provide an introduction to encoding geospatial information in GML 2.0, broken down as follows:

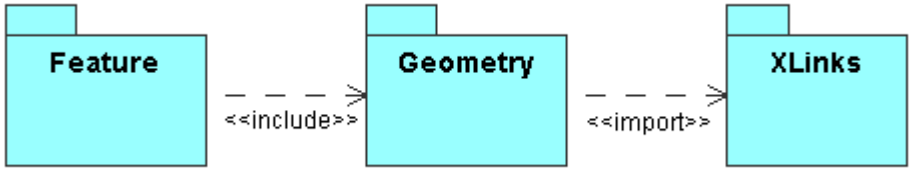
- 4.2 Encoding a feature without geometry
- 4.3 Encoding geometry
- 4.4 Encoding a feature with geometry
- 4.5 Encoding collections of features
- 4.6 Encoding associations between features

First-time readers may wish to skip this sub-section and proceed directly to 4.2; starting from section 4.2 we present an extended set of examples which progessively introduce the components of GML 2.0. Individuals wishing to use GML as a simple means of structuring spatial information for transport are referred to Listing 6.6.

4.1.1 The Geometry schema

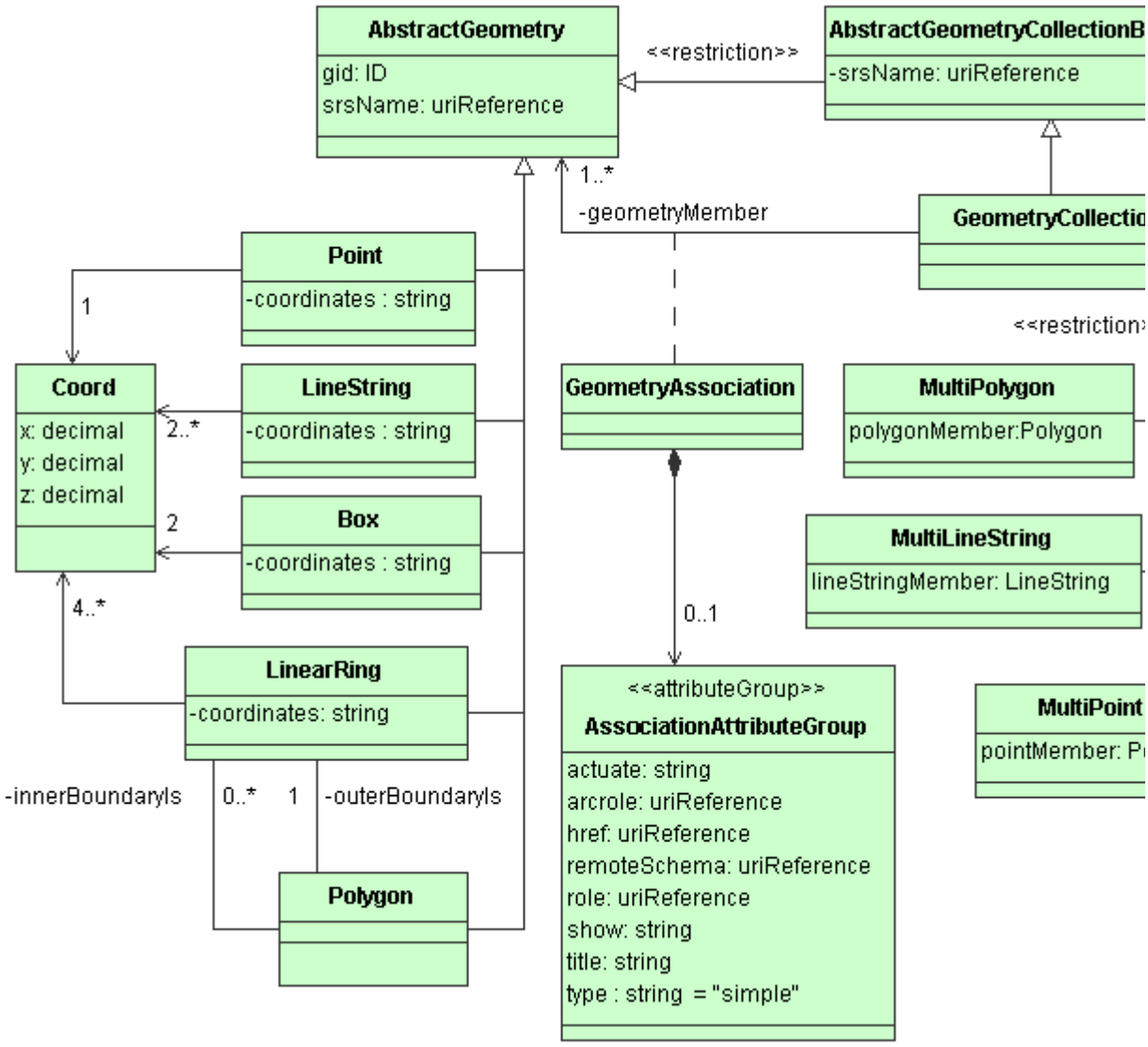
The Unified Modeling Language (UML) offers a fairly general means of visually representing the elements of an application schema; a class diagram presents a concise overview of defined types, and a package diagram depicts higher-level groupings of model elements. The base schemas can be viewed as distinct packages with the dependencies as illustrated in Figure 4.1.

Figure 4.1: Base schemas as packages



The GML Geometry schema includes type definitions for both abstract geometry elements, concrete (multi) point, line and polygon geometry elements, as well as complex type definitions for the underlying geometry types. Figure 4.2 is a UML representation of the Geometry schema; this diagram provides a bridge between the wide-ranging OGC Abstract Specification (Topic 1: *Feature Geometry*) and the GML Geometry schema--it includes many of the 'well-known' structures described in the abstract specification.

Figure 4.2: UML representation of the Geometry schema



The <<restriction>> stereotype applied to a generalization relationship indicates that a subtype defined in the schema is derived by restriction from its supertype. For example, the `MultiLineString` class is a geometry collection in which a member must be a `LineString`. The complete GML Geometry schema appears in Appendix A; it is liberally documented with <annotation> elements. By convention, explicity named type definitions take the corresponding class name and append the 'Type' suffix (e.g. `LineString` becomes `LineStringType`). Type names are in mixed case with a leading capital; the names of geometric properties and attributes are in mixed case with a leading lower case character. The names of abstract elements are in mixed case with a leading underscore (e.g. _Feature) to highlight their abstract character.

The Geometry schema targets the 'gml' namespace. A namespace is a conceptual entity identified by a URI ("http://www.opengis.org/gml" for the core gml namespace) that denotes a

collection of element names and type definitions that belong together--they comprise a cohesive vocabulary. User-defined schemas are required to declare their own target namespace as discussed in section 5.2.4.

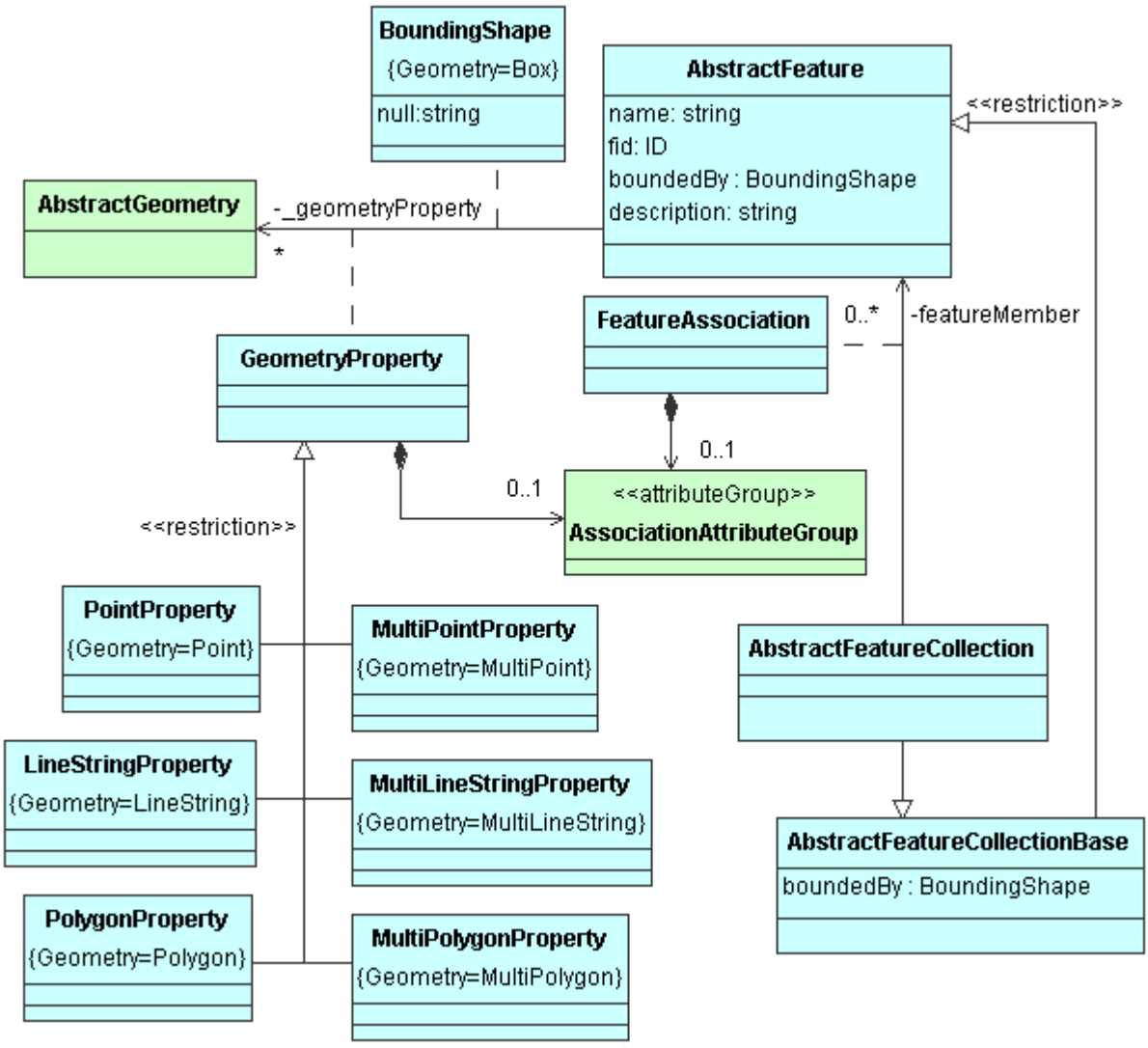
4.1.2 The Feature schema

The Feature schema uses the <include> element to bring in the GML geometry constructs and make them available for use in defining feature types:

```
<include schemaLocation="geometry.xsd"/>
```

Figure 4.3 is a UML representation of the Feature schema. Note that a geometric property is modeled as an association class that links a feature with a geometry; concrete geometric property types such as `PointProperty` constrain the geometry to a particular type (e.g.`Point`).

Figure 4.3: UML representation of the Feature schema



The GML Feature schema is listed in Appendix B. The <include> element in the Feature schema brings in the definitions and declarations contained in the Geometry schema. Like the geometry schema, the Feature schema defines both abstract and concrete elements and types. User-written schemas may define elements and/or types to name and distinguish significant features and feature collections from each other.

4.2 Encoding features without geometry

Although it is not anticipated that many features will be encoded in GML 2.0 without any geometry properties, this section starts off with a simple example based on an aspatial feature. This is referred to as the 'Dean' example. There is a feature type called `Dean` that is defined to have a string property called *familyName* and an integer property called *age*. In addition a `Dean` feature can have zero or more string properties called *nickName*. Thus a single instance

of the Dean feature type might be encoded in XML as:

```
<Dean>
  <familyName>Smith</familyName>
  <age>42</age>
  <nickName>Smithy</nickName>
  <nickName>Bonehead</nickName>
</Dean>
```

Without any thought for GML, one could define the relevant XML Schema to support this as:

```
<element name="Dean" type="ex:DeanType" />
<complexType name="DeanType">
  <sequence>
    <element name="familyName" type="string"/>
    <element name="age" type="integer"/>
    <element name="nickName" type="string" minOccurs="0" maxOccurs="unbounde
  </sequence>
</complexType>
```

However, using the Feature schema from GML, it is necessary to identify what plays the role of features (and their types) and what plays the role of properties. In the Dean example Dean is a feature type and *age* is a property. This is indicated in GML by the following:

```
<element name="Dean" type="ex:DeanType"
  substitutionGroup="gml:_Feature" />

<complexType name="DeanType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="familyName" type="string"/>
        <element name="age" type="integer"/>
        <element name="nickName" type="string" minOccurs="0" maxOccurs="unbo
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

It should be noted that not only does the instance document validate against both of these XML Schema definitions but the content model of DeanType is the same in both XML Schema definitions. Using the Feature schema from GML enables the use of pre-defined attributes. For example features can be identified with the 'fid' attribute, and features can be described using a predefined *gml:description* property. These capabilities are inherited from gml:AbstractFeatureType (to be more precise Dean extends gml:AbstractFeatureType).

```
<Dean fid="D1123">
  <gml:description>A nice old chap</gml:description>
```

```
  <familyName>Smith</familyName>
  <age>42</age>
  <nickName>Smithy</nickName>
  <nickName>Bonehead</nickName>
</Dean>
```

4.3 Encoding geometry

This section describes how GML encodes basic geometry types, and it introduces the GML Geometry schema (geometry.xsd) that supports this encoding. The geometry schema corresponds quite closely to the geometry encoding embodied by the DTD put forth in the GML 1.0 document. The material in this section should be read by all prospective GML users.

In accord with the OGC Simple Features model, GML provides geometry elements corresponding to the following geometry classes:

- Point
- LineString
- LinearRing
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- MultiGeometry

In addition there are <coordinates> and <coord> elements for encoding coordinates, and a <Box> element for defining extents. The following sections describe in detail the encoding of each of these fundamental types of geometry elements.

4.3.1 Coordinates

The coordinates of any Geometry class instance are encoded either as a sequence of <coord> elements that encapsulate tuple components, or as a single string contained within a <coordinates> element. The advantage of using <coord> elements is that a validating XML parser can perform basic type checking and enforce constraints on the number of tuples that appear in a particular geometry instance. Both approaches can convey coordinates in one, two, or three dimensions. The relevant schema fragments can be found in the Geometry schema:

```
<element name="coord" type="gml:CoordType" />

<complexType name="CoordType">
  <sequence>
```

```
    <element name="X" type="decimal"/>
    <element name="Y" type="decimal" minOccurs="0"/>
    <element name="Z" type="decimal" minOccurs="0"/>
  </sequence>
</complexType>
```

An additional level of constraint restricts the number of tuples by data type. For example, a <Point> element contains exactly one coordinate tuple:

```
<Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <coord><X>5.0</X><Y>40.0</Y></coord>
</Point>
```

As an alternative, coordinates can also be conveyed by a single string. By default the coordinates in a tuple are separated by commas, and successive tuples are separated by a space character (#x20). While these delimiters are specified by several attributes, a user is free to define a localized coordinates list that is derived by restriction from `gml:CoordinatesType`. An instance document could then employ the *xsi:type* attribute to substitute the localized coordinates list wherever a <coordinates> element is expected; such a subtype could employ other delimiters to reflect local usage.

It is expected that a specialized client application will extract and validate string content, as these functions will not be performed by a general XML parser. The formatting attributes will assume their default values if they are not specified for a particular instance; the <coordinates> element must conform to these XML Schema fragments:

```
<element name="coordinates" type="gml:CoordinatesType"/>

<complexType name="CoordinatesType">
  <simpleContent>
    <extension base="string">
      <attribute name="decimal" type="string" use="default" value="."/>
      <attribute name="cs" type="string" use="default" value=","/>
      <attribute name="ts" type="string" use="default" value="&#x20;"/>
    </extension>
  </simpleContent>
</complexType>
```

This would allow the Point example provided above to be encoded as:

```
<Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <coordinates>5.0,40.0</coordinates>
</Point>
```

4.3.2 Geometry elements

The coordinates for a geometry are defined within some spatial reference system (SRS), and all geometries must specify this SRS. GML 2.0 does not address the details of defining spatial reference systems. There is currently a proposed XML-based specification for handling coordinate reference systems and coordinate transformations [OGC00-040]. The *srsName* attribute of the geometry types can be used to test for the equality of SRS between different geometries. The *srsName* (since it is a URI reference) may be navigated to the definition of the SRS. It is expected that the pending SRS specification will be applicable to GML encodings, perhaps in the guise of a Geodesy module derived from that specification.

The optional *gid* attribute of the geometry types represents a unique identifier for geometry elements; this is an ID-type attribute whose value must be text string that conforms to all XML name rules (i.e. the first character cannot be a digit).

4.3.3 Primitive geometry elements

The **Point** element is used to encode instances of the Point geometry class. Each <Point> element encloses either a single <coord> element or a <coordinates> element containing exactly one coordinate tuple; the *srsName* attribute is optional since a Point element may be contained in other elements that specify a reference system. Similar considerations apply to the other geometry elements. The Point element, in common with other geometry types, also has an optional *gid* attribute that serves as an identifier. Here's an example:

```
<Point gid="P1" srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <coord><X>56.1</X><Y>0.45</Y></coord>
</Point>
```

The **Box** element is used to encode extents. Each <Box> element encloses either a sequence of two <coord> elements or a <coordinates> element containing exactly two coordinate tuples; the first of these is constructed from the minimum values measured along all axes, and the second is constructed from the maximum values measured along all axes. A value for the *srsName* attribute should be provided, since a Box cannot be contained by other geometry classes. A Box instance looks like this:

```
<Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <coord><X>0.0</X><Y>0.0</Y></coord>
  <coord><X>100.0</X><Y>100.0</Y></coord>
</Box>
```

A **LineString** is a piece-wise linear path defined by a list of coordinates that are assumed to be connected by straight line segments. A closed path is indicated by having coincident first and last coordinates. At least two coordinates are required. Here's an example of a LineString instance:

```
<LineString srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <coord><X>0.0</X><Y>0.0</Y></coord>
  <coord><X>20.0</X><Y>35.0</Y></coord>
  <coord><X>100.0</X><Y>100.0</Y></coord>
</LineString>
```

A **LinearRing** is a closed, simple piece-wise linear path which is defined by a list of coordinates that are assumed to be connected by straight line segments. The last coordinate must be coincident with the first coordinate and at least four coordinates are required (the three to define a ring plus the fourth duplicated one). Since a LinearRing is used in the construction of Polygons (which specify their own SRS), the *srsName* attribute is not needed.

A **Polygon** is a connected surface. Any pair of points in the polygon can be connected to one another by a path. The boundary of the Polygon is a set of LinearRings. We distinguish the outer (exterior) boundary and the inner (interior) boundaries; the LinearRings of the interior boundary cannot cross one another and cannot be contained within one another. There must be at most one exterior boundary and zero or more interior boundary elements. The ordering of LinearRings and whether they form clockwise or anti-clockwise paths is not important. A following example of a Polygon instance has two inner boundaries and uses coordinate strings:

```
<Polygon gid="_98217" srsName="http://www.opengis.net/gml/srs/epsg.xml#4326'
  <outerBoundaryIs>
    <LinearRing>
      <coordinates>0.0,0.0 100.0,0.0 100.0,100.0 0.0,100.0 0.0,0.0</coordina
    </LinearRing>
  </outerBoundaryIs>
  <innerBoundaryIs>
    <LinearRing>
      <coordinates>10.0,10.0 10.0,40.0 40.0,40.0 40.0,10.0 10.0,10.0</coordi
    </LinearRing>
  </innerBoundaryIs>
  <innerBoundaryIs>
    <LinearRing>
      <coordinates>60.0,60.0 60.0,90.0 90.0,90.0 90.0,60.0 60.0,60.0</coordi
    </LinearRing>
  </innerBoundaryIs>
</Polygon>
```

4.3.4 Geometry collections

There are a number of homogeneous geometry collections that are predefined in the Geometry schema. A **MultiPoint** is a collection of Points; a **MultiLineString** is a collection of LineStrings; and a **MultiPolygon** is a collection of Polygons. All of these collections each use an appropriate membership property to contain elements. It should be noted that the *srsName* attribute can only occur on the outermost GeometryCollection and must not appear as an

attribute of any of the enclosed geometry elements. Here's an example of a MultiLineString instance with three members:

```
<MultiLineString srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <lineStringMember>
    <LineString>
      <coord><X>56.1</X><Y>0.45</Y></coord>
      <coord><X>67.23</X><Y>0.98</Y></coord>
    </LineString>
  </lineStringMember>
  <lineStringMember>
    <LineString>
      <coord><X>46.71</X><Y>9.25</Y></coord>
      <coord><X>56.88</X><Y>10.44</Y></coord>
    </LineString>
  </lineStringMember>
  <lineStringMember>
    <LineString>
      <coord><X>324.1</X><Y>219.7</Y></coord>
      <coord><X>0.45</X><Y>4.56</Y></coord>
    </LineString>
  </lineStringMember>
</MultiLineString>
```

In addition the Geometry schema defines a heterogeneous geometry collection represented by the **MultiGeometry** element that provides a container for arbitrary geometry elements; it might contain any of the primitive geometry elements such as Points, LineStrings, Polygons, MultiPoints, MultiLineStrings, MultiPolygons and even other GeometryCollections. The MultiGeometry element has a generic *geometryMember* property which returns the next geometry element in the collection. An example of a heterogeneous MultiGeometry instance appears below.

```
<MultiGeometry gid="c731" srsName="http://www.opengis.net/gml/srs/epsg.xml#4
  <geometryMember>
    <Point gid="P6776">
      <coord><X>50.0</X><Y>50.0</Y></coord>
    </Point>
  </geometryMember>

  <geometryMember>
    <LineString gid="L21216">
      <coord><X>0.0</X><Y>0.0</Y></coord>
      <coord><X>0.0</X><Y>50.0</Y></coord>
      <coord><X>100.0</X><Y>50.0</Y></coord>
    </LineString>
  </geometryMember>

  <geometryMember>
    <Polygon gid="_877789">
      <outerBoundaryIs>
        <LinearRing>
```

```
          <coordinates>0.0,0.0 100.0,0.0 50.0,100.0 0.0,0.0</coordinates>
        </LinearRing>
      </outerBoundaryIs>
    </Polygon>
  </geometryMember>
</MultiGeometry>
```

4.4 Encoding features with geometry

GML 2.0 provides a pre-defined set of geometry properties that can be used to relate geometries of particular types to features. Consider the case where the `DeanType` feature definition has a point property called *location*, which is one of the pre-defined descriptive names that can substitute for the formal name *pointProperty*.

```
<Dean>
  <familyName>Smith</familyName>
  <age>42</age>
  <nickName>Smithy</nickName>
  <nickName>Bonehead</nickName>
  <gml:location>
    <gml:Point>
      <gml:coord><gml:X>1.0</gml:X><gml:Y>1.0</gml:Y></gml:coord>
    </gml:Point>
  <gml:location>
</Dean>
```

which is based on the following application schema fragment:

```
<element name="Dean" type="ex:DeanType" substitutionGroup="gml:_Feature"/>

<complexType name="DeanType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="familyName" type="string"/>
        <element name="age" type="integer"/>
        <element name="nickName" type="string" minOccurs="0"
          maxOccurs="unbounded"/>
        <element ref="gml:location"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Alternatively one can define geometry properties specific to an application schema. For example one might wish to name the property specifying the location of the `Dean` instance as *deanLocation*:

```
<element name="Dean" type="ex:DeanType" substitutionGroup="gml:_Feature"/>

<complexType name="DeanType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="familyName" type="string"/>
        <element name="age" type="integer"/>
        <element name="nickName" type="string" minOccurs="0"
          maxOccurs="unbounded"/>
        <element name="deanLocation" type="gml:PointPropertyType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

In this example `gml:PointPropertyType` is available as a useful pre-defined property type for a feature to employ in just the same way that strings and integers are. The local declaration of the <deanLocation> element basically establishes an alias for <gml:pointProperty> as a subelement of `Dean`.

The exclusive use of globally-scoped element declarations reflects a different authoring style that 'pools' all elements in the same symbol space (see section 2.5 of the XML Schema specification, Part 1 for further details); this style also allows us to assign elements to a substitution group such that designated elements can substitute for a particular *head element*, which must be declared as a global element. The *deanLocation* property would be declared globally and referenced in a type definition as shown below:

```
<element name="Dean" type="ex:DeanType" substitutionGroup="gml:_Feature"/>
<element name="deanLocation" type="gml:PointPropertyType"
  substitutionGroup="gml:pointProperty"/>

<complexType name="DeanType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="familyName" type="string"/>
        <element name="age" type="integer"/>
        <element name="nickName" type="string" minOccurs="0" maxOccurs="unbo
        <element ref="ex:deanLocation" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

4.5 Encoding feature collections

GML 2.0 provides support for building feature collections. An element in an application

schema that plays the role of a feature collection must derive from `gml:AbstractFeatureCollectionType` and declare that it can substitute for the (abstract) <gml:_FeatureCollection> element. A feature collection can use the *featureMember* property to show containment of other features and/or feature collections.

Consider the 'Cambridge' example (described in Section 5) where a `CityModel` feature collection contains `Road` and `River` feature members. In this modification to the example the features are contained within the `CityModel` using the generic *gml:featureMember* property that instantiates the `gml:FeatureAssociationType`:

```
<CityModel fid="Cm1456">
  <dateCreated>Feb 2000</dateCreated>
  <gml:featureMember>
    <River fid="Rv567">....</River>
  </gml:featureMember>

  <gml:featureMember>
    <River fid="Rv568">....</River>
  </gml:featureMember>

  <gml:featureMember>
    <Road fid="Rd812">....</Road>
  </gml:featureMember>
</CityModel>
```

With the following associated application schema fragments:

```
<element name="CityModel" type="ex:CityModelType"
  substitutionGroup="gml:_FeatureCollection"/>
<element name="River" type="ex:RiverType" substitutionGroup="gml:_Feature"/>
<element name="Road" type="ex:RoadType" substitutionGroup="gml:_Feature"/>

<complexType name="CityModelType">
  <complexContent>
    <extension base="gml:AbstractFeatureCollectionType">
      <sequence>
        <element name="dateCreated" type="month"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="RiverType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>....</sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="RoadType">
```

```
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>.....</sequence>
    </extension>
  </complexContent>
</complexType>
```

GML 2.0 provides a mechanism for feature identification. All GML features have an optional 'fid' attribute of type ID inherited from `gml:AbstractFeatureType`; this means that features in the same GML document cannot share a 'fid' value. The 'fid' attribute and simple link elements constructed using XLink attributes provide a means of unambiguously referencing specific features within a GML document.

Within a feature collection, a <featureMember> element can either contain a single feature or point to a feature that is stored remotely (including elsewhere in the same document). A simple link element can be constructed by including a specific set of XLink attributes. The XML Linking Language (XLink) is currently a Proposed Recommendation of the World Wide Web Consortium [XLink]. XLink allows elements to be inserted into XML documents so as to create sophisticated links between resources; such links can be used to reference remote properties.

A simple link element can be used to implement pointer functionality, and this functionality has been built into various GML 2.0 elements by including the `gml:AssociationAttributeGroup` in these constructs:

- gml:FeatureAssociationType,
- the geometry collection types, and
- all of the pre-defined geometry property types.

As an example, we can modify the `CityModel` fragment shown above to include a remote `River` members *without* making any changes to the application schema.

```
<CityModel fid="Cm1456">
  <dateCreated>Feb 2000</dateCreated>

  <gml:featureMember xlink:type="simple"
    xlink:href="http://www.myfavoritesite.com/rivers.xml#Rv567"/>

  <gml:featureMember xlink:type="simple"
    xlink:href="http://www.myfavoritesite.com/rivers.xml#Rv568"/>

  <gml:featureMember>
    <Road fid="Rd812">....</Road>
  </gml:featureMember>
</CityModel>
```

It should be noted that the *featureMember* property can both point to a remote feature *and* contain a feature. It is not possible in XML Schema to preclude this practice using a purely grammar-based validation approach. The GML 2.0 specification regards a *featureMember* in this state to be undefined.

The most basic syntax for a simple link is as follows:

```
<propertyName xlink:type="simple"
  xlink:title="Description of target instance"
  xlink:href="http://www.myfavoritesite.com/locations.xml#identifier" />
```

where the xlink:title attribute is optional. The xlink:href attribute must point to an object whose type matches that of the value type of the property. It is up to the application to validate that this is the case, an XML parser would not place any constraints on the element linked to by the href attribute.

To enhance clarity a new attribute defined in the 'gml' namespace is introduced to supplement the basic XLink attributes: *remoteSchema*. The *remoteSchema* attribute is a URI reference that uses XPointer syntax to identify the GML schema fragment that constrains the resource pointed to by the locator attribute; this additional attribute is included in the `gml:AssociationAttributeGroup` and so is already available to *featureMember* properties so that they can be expressed like this (assuming "RiverType" is the value of some identifier):

```
<gml:featureMember xlink:type="simple"
  gml:remoteSchema="http://www.myfavoritesite.com/types.xsd#RiverType"
  xlink:href="http://www.myfavoritesite.com/rivers#Rv567"/>
```

XLink attributes can **only** be placed on property elements, and there are no constraints on the values of the xlink:title attribute. Simple XLinks also allow the use of the xlink:role attribute. However this is most commonly a reflection of the property (for example the *featureMember* role name) that is using the link.

The XLink specification requires that the xlink:href attribute point to the resource participating in the link by providing a Uniform Resource Identifier (URI). For example, such a URI may constitute a request to a remote server such as an OGC Web Feature Server (WFS). It might be noted that in response to a 'GETFEATURE' request a WFS will return the GML description of a feature, provided the feature identifier (the value of its 'fid' attribute) is known:

```
xlink:href="http://www.myfavoritesite.com/wfs?WFSVER=0.0.12
  &amp;REQUEST=GETFEATURE
  &amp;FEATUREID=Rv567"
```

The value of the xlink:href **must** be a valid URI per IETF RFC 2396 [RFC2396] and RFC

2732 [RFC2732]; as a consequence, certain characters in the URI string must be escaped according to the URI encoding rules set forth in the XLink specification and in the aforementioned IETF documents.

It might be noted that the WFS 'GETFEATURE' request returns a single feature. If the *gml:remoteSchema* attribute is being used, then it should point to the definition of the relevant feature type. Alternatively an XLink can be used to encode an entire query request to a WFS (required character references do not appear in the query string for clarity):

```
xlink:href="http://www.myfavoritesite.com/wfs?
  WFSVER=0.0.12&
  REQUEST=GETFEATURE&
  TYPENAME=INWATERA_1M&
  FILTER='<Filter>
            <Not>
              <Disjoint>
                <PropertyName>INWATERA_1M.WKB_GEOM</PropertyName>
                <Polygon srsName="http://www.opengis.net/gml/srs/epsg.xml#43
                  <outerboundaryIs>
                    <LinearRing>
                      <coordinates>
                        -150,50 -150,60 -125,60 -125,50 -150,50
                      </coordinates>
                    </LinearRing>
                  </outerboundaryIs>
                </Polygon>
              </Disjoint>
            </Not>
          </Filter>'"
```

In this case the WFS may 'manufacture' a generic feature collection to hold the results of the query. But since feature collections also have a feature type, the *gml:remoteSchema* attribute (if used) should point to the feature type definition of this feature collection, not the types of the features returned by the query.

4.6 Encoding feature associations

The essential purpose of XML document structure is to describe data and the relationships between components within it. With GML it is possible to denote relationships either by containment (binary relationships only) or by linking, but there is no *a priori* reason for preferring one style over another. The examples presented so far have emphasized containment, generally using the property name (i.e. the role name) as a 'wrapper' to make the logical structure explicit.

The GML data model is based on three basic constructs: Feature, Feature Type (i.e. Class), and Property. Both Classes and Properties are resources that are independently defined in

GML. A property is defined by specifying the range and domain of the property, each of which must be defined GML types. A GML feature is a Class on which are defined a number of properties; zero, one, or more of these properties are geometry-valued. A *Resource* is anything that has an identity [RFC2396]; this can include electronic documents, a person, a service or a geographic feature.

In the previous section specialized links were used to allow a feature collection in a GML document to contain features external to the document. Had the features been present in the GML document, there would have been no compelling reason to use a link, since the 'containment' relationship could have been indicated by using nesting in the GML document. However there are many relationships that cannot be encoded using containment. Consider the 'adjacency' relationship between three `LandParcel` features. In total there are three relationships representing the three pairings of features. Using the nesting approach encouraged so far, one might be tempted to encode this in the following, albeit messy, way using an *adjacentTo* property (we will present a much cleaner form later):

```
<LandParcel fid="Lp2034">
  <area>2345</area>
  <gml:extentOf>...</extentOf>
  <adjacentTo>
    <LandParcel fid="Lp2035">
      <area>9812</area>
      <gml:extentOf>...</extentOf>
      <adjacentTo xlink:type="simple" xlink:href="#Lp2034"/>
      <adjacentTo>
        <LandParcel fid="Lp2036">
          <area>8345</area>
          <gml:extentOf>...</extentOf>
          <adjacentTo xlink:type="simple" xlink:href="#Lp2034"/>
          <adjacentTo xlink:type="simple" xlink:href="#Lp2035"/>
        </LandParcel>
      </adjacentTo>
    </LandParcel>
  </adjacentTo>
  <adjacentTo xlink:type="simple" xlink:href="#Lp2036"/>
</LandParcel>
```

In the above fragment the links are being used to identify features within the same GML document. In this example the adjacency relationship is binary (it connects pairs of features) and bi-directional (it can be navigated in both directions). This has been achieved by using two *adjacentTo* properties (each with a simple XLink) to represent each relationship. The relationship itself has no identity in this encoding, and it is not possible to record properties on the relationship. Relationships with these characteristics are sometimes referred to as 'lightweight' relationships.

In the above encoding the *adjacentTo* property sometimes 'nests' the related feature and sometimes 'points' to it. A more symmetrical version of the above would be:

```
<LandParcel fid="Lp2034">
  <area>2345</area>
  <gml:extentOf>...</extentOf>
  <adjacentTo xlink:type="simple" xlink:href="#Lp2035"/>
  <adjacentTo xlink:type="simple" xlink:href="#Lp2036"/>
</LandParcel>
....
<LandParcel fid="Lp2035">
  <area>9812</area>
  <gml:extentOf>...</extentOf>
  <adjacentTo xlink:type="simple" xlink:href="#Lp2034"/>
  <adjacentTo xlink:type="simple" xlink:href="#Lp2036"/>
</LandParcel>
....
<LandParcel fid="Lp2036">
  <area>8345</area>
  <gml:extentOf>...</extentOf>
  <adjacentTo xlink:type="simple" xlink:href="#Lp2034"/>
  <adjacentTo xlink:type="simple" xlink:href="#Lp2035"/>
</LandParcel>
```

Here the ellipses represent the necessary containment constructs holding this set of `LandParcels` in some root feature collection. However both examples conform to the same application schema:

```
<element name="LandParcel" type="ex:LandParcelType"
  substitutionGroup="gml:_Feature"/>
<element name="adjacentTo" type="ex:AdjacentToType"
  substitutionGroup="gml:featureMember" />

<complexType name="LandParcelType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="area" type="integer"/>
        <element ref="gml:extentOf"/>
        <element ref="ex:adjacentTo" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="AdjacentToType">
  <complexContent>
    <restriction base="gml:FeatureAssociationType">
      <sequence>
        <element ref="ex:LandParcel"/>
      </sequence>
    </restriction>
  </complexContent>
</complexType>
```

A 'heavyweight' relationship provides identity for the relationship and an opportunity to have

properties on the relationship. To extend the adjacency example, one might wish to have a heavyweight adjacency relationship represented by an `AdjacentPair` feature type with the length of the common border recorded as a *commonBoundaryLength* property. This might be encoded in GML as:

```
<LandParcel fid="Lp2034">
  <area>2345</area>
  <gml:extentOf>...</extentOf>
</LandParcel>
....
<LandParcel fid="Lp2035">
  <area>9812</area>
  <gml:extentOf>...</extentOf>
</LandParcel>
....
<AdjacentPair fid="Ad1465">
  <commonBoundaryLength>231</commonBoundaryLength>
  <adjacentTo xlink:type="simple" xlink:href="#Lp2034"/>
  <adjacentTo xlink:type="simple" xlink:href="#Lp2035"/>
</Adjacent>
```

Note that in this example the `AdjacentPair` instances could exist in a separate GML document from the `LandParcel` features since the *adjacentTo* properties can point to features outside of the GML document. Whether the instances are in a single or multiple documents, they all conform to the same application schema:

```
<element name="LandParcel" type="ex:LandParcelType"
  substitutionGroup="gml:_Feature"/>
<element name="adjacentTo" type="ex:AdjacentToType"
  substitutionGroup="gml:featureMember"/>
<element name="AdjacentPair" type="ex:AdjacentPairType"
  substitutionGroup="gml:_Feature"/>

<complexType name="LandParcelType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="area" type="integer"/>
        <element ref="gml:extentOf"/>
        <!-- note adjacentTo has been removed -->
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="AdjacentToType">
  <complexContent>
    <restriction base="gml:FeatureAssociationType">
      <sequence>
        <element ref="ex:LandParcel"/>
      </sequence>
    </restriction>
```

```
  </complexContent>
</complexType>

<complexType name="AdjacentPairType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="commonBoundaryLength" type="integer"/>
        <element ref="ex:adjacentTo" minOccurs="2" maxOccurs="2"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

The use of extended-type XLink elements offers a more concise means of handling n-ary associations and support for creating third-party linkbases. However, such topics are beyond the scope of this document and interested readers are encouraged to consult the XLink specification for details.

5 GML application schemas

6: Worked examples Go

5.1 Introduction

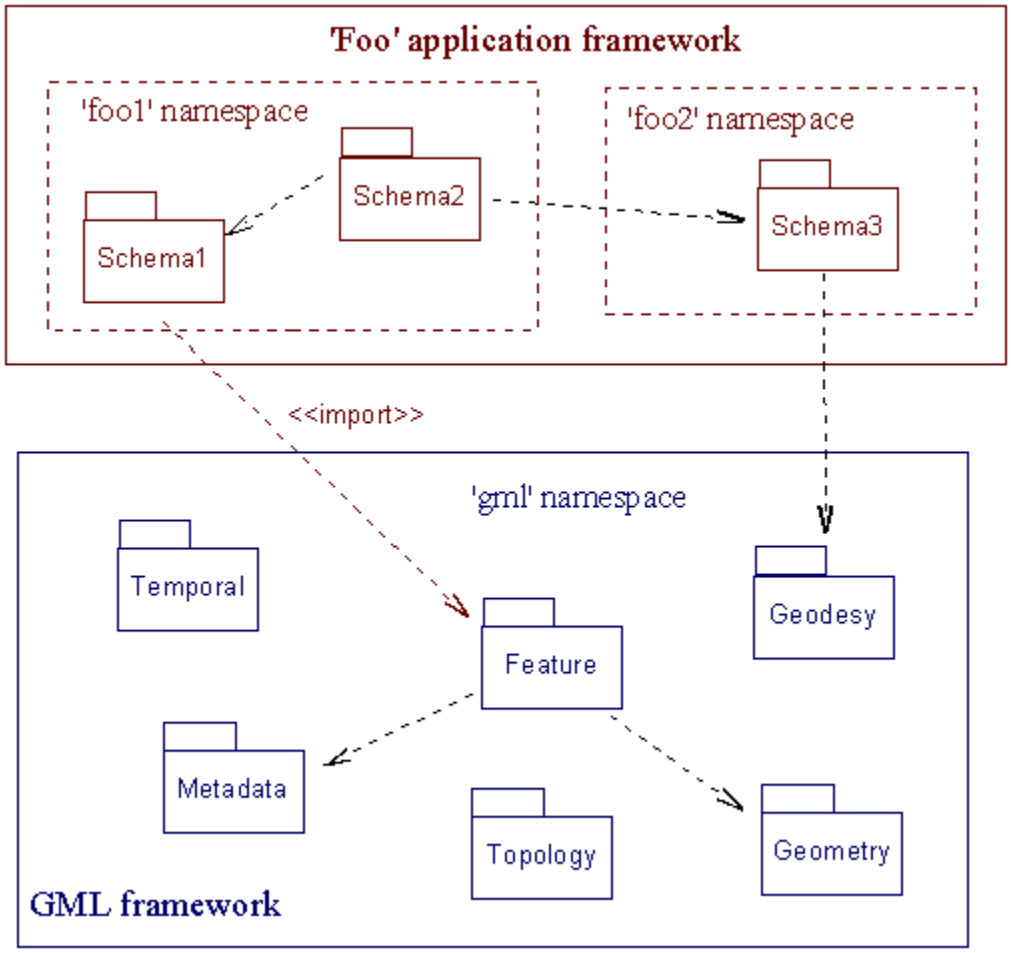
The base schemas (Geometry, Feature, XLink) can be viewed as the components of an application framework for developing schemas or sets of schemas that pertain to a particular domain (e.g. Forestry), jurisdiction (e.g. France), or information community. Furthermore, such application schemas may be developed in a more horizontal fashion to support many information communities.

There are some basic conformance requirements that every application schema must satisfy. Specifically, a conforming GML application schema must heed the following general requirements:

1. an application schema must adhere to the detailed schema development rules described in Section 5.2
2. an application schema must not change the name, definition, or data type of mandatory GML elements.
3. abstract type definitions may be freely extended or restricted.
4. the application schema must be made available to anyone receiving data structured according to that schema.
5. the relevant schemas must specify a target namespace that *must not* be `http://www.opengis.net/gml` (i.e. the 'gml' namespace).

A set of logically-related GML schemas, which we term the GML Framework, is depicted in Figure 5.1. The GML schemas provide basic constructs for handling geospatial data in a modular manner. A more specialized application framework containing component application schemas would typically be created for a particular theme or domain, but may also be quite horizontal in nature.

Figure 5.1: GML as a core framework



5.2 Rules for constructing application schemas (normative)

The following rules must be adhered to when constructing GML application schemas.

5.2.1 Defining new feature types

Developers of application schemas can create their own feature or feature collection types, but they must ensure that these concrete feature and feature collection types are subtyped (either

directly or indirectly) from the corresponding GML types: `gml:AbstractFeatureType` or `gml:AbstractFeatureCollectionType`.

```
<complexType name="MyFeature1Type">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <!-- additional child elements inserted here -->
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="MyFeature2Type">
  <complexContent>
    <extension base="foo:MyFeature1Type">
      <sequence>
        <!-- additional child elements inserted here -->
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

5.2.2 Defining new geometry types

Authors may create their own geometry types if GML lacks the desired construct. To do this, authors must ensure that these concrete geometry and geometry collection types are subtyped (either directly or indirectly) from the corresponding GML types: `AbstractGeometryType` or `GeometryCollectionType`:

```
<complexType name="MyGeometry1Type">
  <complexContent>
    <extension base="gml:AbstractGeometryType">
      <sequence>
        <!-- additional child elements inserted here -->
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Any user-defined geometry subtypes shall inherit the elements and attributes of the base GML geometry types without restriction, but may extend these base types to meet application requirements, such as providing a finer degree of interoperability with legacy systems and data sets.

5.2.3 Defining new geometry properties

Authors may create their own geometry properties that encapsulate geometry types they have defined according to subsection 5.2.2; they must ensure that these properties are subtyped (either directly or indirectly) from `gml:GeometryPropertyType` and that they do not change the cardinality of the target instance, which must be a bonafide geometry construct:

```
<complexType name="MyGeometry1PropertyType">
  <complexContent>
    <restriction base="gml:GeometryPropertyType">
      <sequence minOccurs="0">
        <element ref="foo:MyGeometry1Type" />
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>
```

5.2.4 Declaring a target namespace

Authors must declare a target namespace for their schemas. All elements declared in the schema, along with their type definitions, will reside in this namespace. Validation will not succeed if the instance document does not reside in the schema's target namespace. Note that it is not a requirement that URIs actually point to anything concrete, such as a schema document; namespaces are basically just a mechanism to keep element names distinct, thereby preventing namespace 'collisions'.

To use namespaces, elements are given qualified names (QName) that consist of two parts: the *prefix* is mapped to a URI reference and signifies the namespace to which the element belongs; the *local part* conforms to the usual NCName production rules from the W3C Namespaces Recommendation:

```
NCName ::=  (Letter | '_') (NCNameChar)*
NCNameChar ::=  Letter | Digit | '.' | '-' | '_' | CombiningChar | Extender
```

In each worked example the namespace for all elements is explicitly indicated in order to show how vocabularies from different namespaces can intermingle in a single instance document. The use of fully qualified names is specified by setting the value of the *elementFormDefault* attribute of <schema> to "qualified":

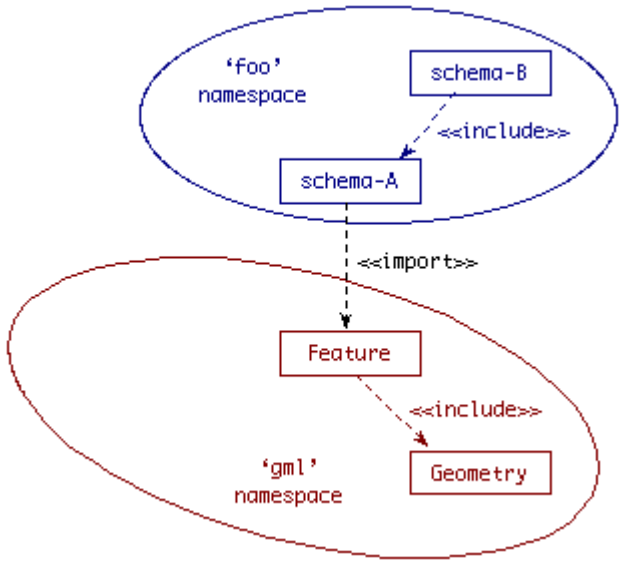
```
<schema  targetNamespace="http://www.bar.net/foo"
         xmlns="http://www.w3.org/2000/10/XMLSchema"
         xmlns:gml="http://www.opengis.net/gml"
         xmlns:foo="http://www.bar.net/foo"
         elementFormDefault="qualified"
         version="0.1">

  <!-- import constructs from the GML Feature and Geometry schemas -->
```

```
  <import namespace="http://www.opengis.net/gml" schemaLocation="feature.xsd
 . . .
</schema>
```

5.2.5 Importing schemas

A conforming instance document can employ constructs from multiple namespaces, as indicated in Figure 5.2. Schema-A in the 'foo' namespace plugs into the GML framework via the Feature schema (which also brings along the geometry constructs). The Feature schema resides in the 'gml' namespace along with the Geometry schema, so it uses the *include* mechanism. However, since Schema-A targets a different namespace, it must employ the *import* mechanism to use the core GML constructs.

Figure 5.2: Using schemas from multiple namespaces



Authors must bear in mind that the 'import' mechanism is the only means whereby an application schema can bring in GML constructs and make use of them. Since a conforming application schema cannot target the core GML namespace ("http://www.opengis.net/gml"), other mechanisms such as 'include' and 'redefine' cannot be employed.

5.2.6 Using substitution groups

Global elements that define substitution groups shall be used for both class (e.g. feature, geometry etc.) and properties that are defined in GML application schemas, and can substitute for the corresponding GML elements wherever these are expected. Declare an element globally and specify a suitable substitution group if the element is required to substitute for

another (possibly abstract) element; substitution groups thus enable type promotion (i.e. treating a specific type as a more general supertype). The following global declaration ensures that if `foo:CircleType` is a defined geometry type, then a <Circle> element can appear wherever the (abstract) gml:_Geometry element is expected:

```
<schema . . .>
  <element name="Circle" type="foo:CircleType"
    substitutionGroup="gml:_Geometry" />
  . . .
</schema>
```

Identical elements declared in more than one complex type definition in a schema should reference a global element. If the <Circle> element is declared globally in the 'foo' namespace (as shown above), it is referenced from within a type definition as follows:

```
<complexType name="MyHubPropertyType">
  <complexContent>
    <restriction base="gml:GeometryPropertyType">
      <sequence minOccurs="0">
        <element ref="foo:Circle" />
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>
```

5.2.7 Declaring additional properties

Schema authors can use the built-in geometric properties or derive their own when necessary as shown in subsection 5.2.3. GML provides a number of predefined geometric properties: *location*, *centerLineOf*, *extentOf*, and so on. Authors can also apply a different name to a base type and use it instead:

```
<element name="crashSite" type="gml:PointPropertyType"
  substitutionGroup="gml:pointProperty" />
```

A feature may also have properties that possess a more complex content model (like geometry properties or feature members). It's important to keep in mind that complex properties represent binary relationships, and that property elements carry the role name of that association. In general, the value of a property element must be one or more instances (either local or remote) of some defined simple or complex type.

As an example, consider the *inspectedBy* property of a waste handling facility that associates a <WasteFacility> instance with an <Inspector> instance:

```
<complexType name="WasteFacilityType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <!-- other properties -->
        <element name="inspectedBy" type="foo:InspectedByType" />
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="InspectedByType">
  <sequence minOccurs="0">
    <element ref="foo:Inspector"/>
  </sequence>
  <attributeGroup ref="gml:AssociationAttributeGroup" />
</complexType>
```

The above fragment indicates that `foo:InspectionType` includes the simple-type XLink attributes (that are bundled up in `gml:AssociationAttributeGroup`) to take advantage of the optional pointer functionality; this can be useful if the same inspector assesses multiple facilities and is the target of multiple references:

```
<WasteFacility>
  . . .
  <inspectedBy xlink:type="simple" xlink:href="inspectors.xml#n124" />
</WasteFacility>
```

The definition of the `Inspector` type is not shown here, but it must appear in some application schema. Furthermore, an inspector need not be a feature instance--perhaps an inspector is a `Person` that possesses properties such as an 'oid' identifier. Without the *href* and other XLink attributes a valid instance might then look like the following:

```
<WasteFacility>
  . . .
  <inspectedBy>
    <Inspector oid="n124">. . .</Inspector>
  </inspectedBy>
</WasteFacility>
```

5.2.8 Defining new feature association types

Developers of application schemas can create their own feature association types that must derive from `gml:FeatureAssociationType`. The target instance must be a bonafide GML feature, and it may appear once (explicitly minOccurs="0", implicitly maxOccurs="1"):

```
<complexType name="MyFeatureAssociationType">
  <complexContent>
    <restriction base="gml:FeatureAssociationType">
      <sequence minOccurs="0">
        <element ref="foo:MyFeature1Type" />
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>
```

In many cases it may be desirable to allow instances of only certain feature types as members of a feature collection. A *Feature Filter* as described below shall be applied to ensure that only properly labeled features are valid members.

Feature Filter

Intent:

Restrict membership in a feature collection to permit only instances of specified feature types as allowable members.

Also Known As:

Barbarians at the Gate

Motivation:

Feature collections that extend `gml:AbstractFeatureCollectionType` are somewhat 'promiscuous' in that they will accept any concrete GML feature as an allowable member. A "Feature Filter" can be applied to ensure that only properly labeled features are valid members.

Implementation:

1. Declare a set of abstract elements to 'label' allowable members in a feature collection:

   ```
   <element name="_SchoolFeature" type="gml:AbstractFeatureType"
     substitutionGroup="gml:_Feature" abstract="true"/>
   ```

2. Define a filter by restricting `gml:FeatureAssociationType`:

   ```
   <complexType name="SchoolMemberType">
     <complexContent>
       <restriction base="gml:FeatureAssociationType">
         <sequence minOccurs="0">
           <element ref="ex:_SchoolFeature"/>
         </sequence>
         <attributeGroup ref="gml:AssociationAttributeGroup"/>
       </restriction>
     </complexContent>
   </complexType>
   ```

3. Label allowable features as they are declared globally:

```
<element name="School" type="ex:SchoolType"
  substitutionGroup="ex:_SchoolFeature"/>
```

6 Worked examples of application schemas (non-normative)

7: Profiles of GML Go

This section presents several examples to illustrate the construction of application schema that employ the base GML schemas. All examples in this document have been validated using the XSV parser (version 1.166/1.77, 2000-09-28), a Python implementation that supports the current XML Schema Candidate Recommendation; several other parsers (both commercial and open-source implementations) now also solidly support the current specification. An online validation service that uses XSV is available at this URL: <http://www.w3.org/2000/09/webdata/xsv>; while this checking service requires that all documents be web-accessible, it is also possible to download the source files and use the parser offline.

Two examples are briefly summarized here and are explored in some depth using UML class diagrams, corresponding GML application schemas, and sample instance documents. One example is based on a simple model of the city of Cambridge, and is described more fully in Box 1.1 below.

Box 1.1: The Cambridge example

This example has a single feature collection of type `CityModel` and contains two features using a containment relationship called 'cityMember'. The feature collection has a string property called *dateCreated* with the value 'Feb 2000' and a geometric property called *boundedBy* with a 'Box' value. The Box geometry (which represents the 'bounding box' of the feature collection) is expressed in the SRS identified by the value of the *srsName* attribute: this URI reference points to a fragment in another XML document that contains information about the reference system.

The first feature member is an instance of `RiverType` with the name "Cam" and description "The river that runs through Cambridge"; it has a geometric property called *centerLineOf* with a LineString value. The LineString geometry is expressed in the same SRS used by the bounding box.

The second feature member is an instance of `RoadType` with description "M11". It has a string property called *classification* with value "motorway" and an integer property called *number* with value "11". The road has a geometric property called *linearGeometry* with a

LineString value; this LineString geometry is also expressed in the same SRS used by the bounding box.

In the 'Cambridge' example the first feature member uses only standard property names defined by GML, whereas the second feature member uses application-specific property names. Thus this example will demonstrate how GML is capable of being used in a custom application model, but it is not intended to provide examples of how the various types of geometry are encoded.

A second example will be used to illustrate how GML can represent a hierarchy of feature collections; this will be referred to as the 'Schools' example and it is summarized in Box 1.2.

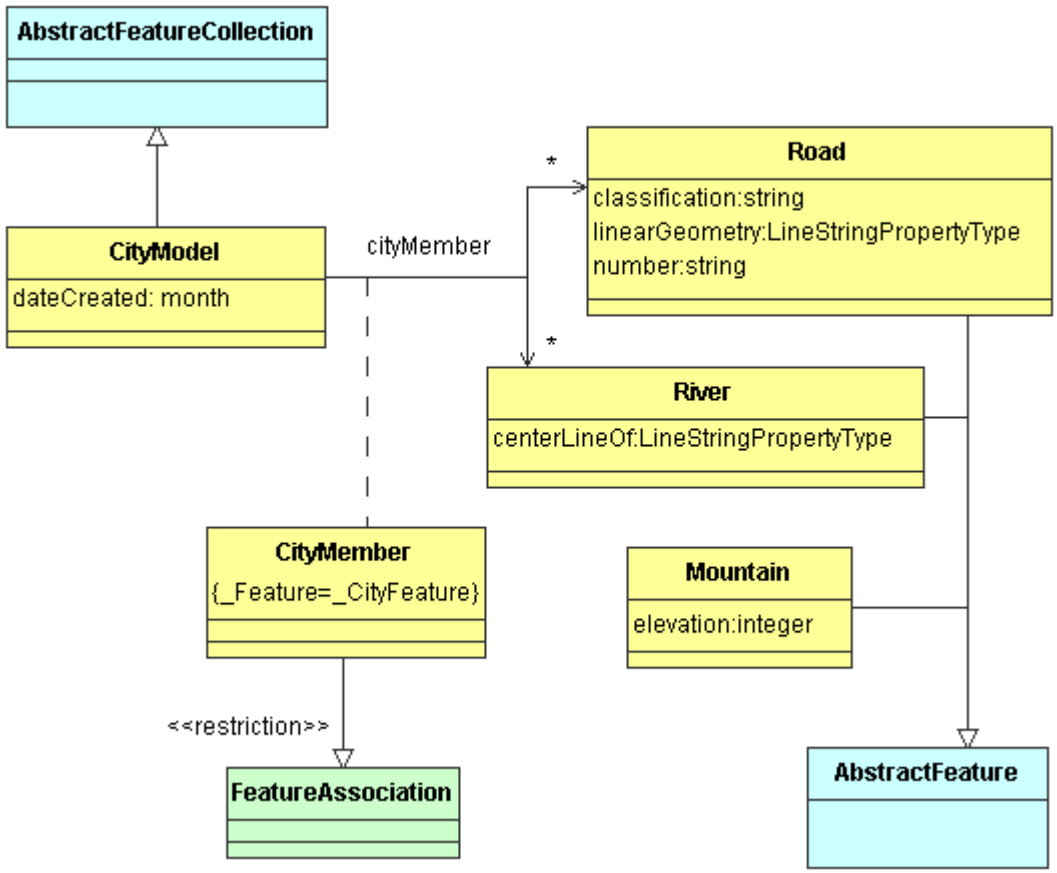
Box 1.2: The Schools example

In this example we have a root feature collection of type `StateType` that contains two features collections (instances of `SchoolDistrictType`) using the pre-defined containment relationship 'featureMember'. The State collection also has a *studentPopulation* property. Each of the `SchoolDistrict` collections contains two School or College features using the containment relationship 'schoolMember'.

A `SchoolDistrict` feature has a string property called *name* and a polygon property called *extentOf*. A `School` feature has a string property called *address* and a point property called *location*. A `College` feature also has a string property called *address* plus a point property called *pointProperty*.

Figure 6.1 is a UML diagram for the Cambridge example. As shown, allowable city members must be `Road` or `River` instances; a `Mountain` instance is not a valid member of the feature collection.

Figure 6.1: UML diagram for the Cambridge example



Listing 6.1 is a custom city schema for the Cambridge example. The explicit reference to "city.xsd" in the root element of the instance document in Listing 6.2 (i.e. the value of the *xsi:schemaLocation* attribute) is not required, but in this case it provides a hint to the validating parser regarding the location of a relevant schema document.

Listing 6.1: city.xsd View so

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- File: city.xsd  -->
<schema targetNamespace="http://www.opengis.net/examples"
        xmlns:ex="http://www.opengis.net/examples"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xmlns:gml="http://www.opengis.net/gml"
        xmlns="http://www.w3.org/2000/10/XMLSchema"
        elementFormDefault="qualified"
        version="2.03">

  <annotation>
    <appinfo>city.xsd v2.03 2001-02</appinfo>
    <documentation xml:lang="en">
      GML schema for the Cambridge example
    </documentation>
  </annotation>
```

```
<!-- import constructs from the GML Feature and Geometry schemas -->
<import namespace="http://www.opengis.net/gml" schemaLocation="feature.xs

<!-- ==========================================================
     global element declarations
========================================================== -->

<element name="CityModel" type="ex:CityModelType"
   substitutionGroup="gml:_FeatureCollection" />
<element name="cityMember" type="ex:CityMemberType"
  substitutionGroup="gml:featureMember"/>
<element name="Road" type="ex:RoadType"
  substitutionGroup="ex:_CityFeature"/>
<element name="River" type="ex:RiverType"
  substitutionGroup="ex:_CityFeature"/>
<element name="Mountain" type="ex:MountainType"
  substitutionGroup="gml:_Feature"/>

<!-- a label for restricting membership in the CityModel collection -->
<element name="_CityFeature" type="gml:AbstractFeatureType" abstract="tru
  substitutionGroup="gml:_Feature"/>

<!-- ==========================================================
     type definitions for city model
========================================================== -->

<complexType name="CityModelType">
  <complexContent>
    <extension base="gml:AbstractFeatureCollectionType">
      <sequence>
        <element name="dateCreated" type="month"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="CityMemberType">
  <annotation>
    <documentation>
      A cityMember is restricted to those features (or feature
      collections)that are declared equivalent to ex:_CityFeature.
    </documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:FeatureAssociationType">
      <sequence minOccurs="0">
        <element ref="ex:_CityFeature"/>
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>

<complexType name="RiverType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
```

```
          <element ref="gml:centerLineOf"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="RoadType">
    <complexContent>
      <extension base="gml:AbstractFeatureType">
        <sequence>
          <element name="linearGeometry" type="gml:LineStringPropertyType"/>
          <element name="classification" type="string"/>
          <element name="number" type="string"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- this is just here to demonstrate feature member restriction -->
  <complexType name="MountainType">
    <complexContent>
      <extension base="gml:AbstractFeatureType">
        <sequence>
          <element name="elevation" type="integer"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</schema>
```

Note that the application schema targets the 'ex' namespace; it imports the GML feature and geometry constructs from the 'gml' namespace. The <boundedBy> element is defined in the Feature schema; the <name> and <description> elements are also defined there. The <CityModel> element is an instance of the user-defined `ex:CityModelType` type that is derived by extension from `gml:AbstractFeatureCollectionType`. The types `ex:RiverType` and `ex:RoadType` are both derived by extension from `gml:AbstractFeatureType`, which is defined in the GML Feature schema; these derivations assure that the application schema conforms with the GML implementation specification of the OGC Simple Feature model.

Listing 6.2 is a simple schema-valid instance document that conforms to city.xsd. A few words of explanation about the <Mountain> feature are in order! If this particular cityMember is uncommented in Listing 6.2, it will raise a validation error because even though the mountain is a well-formed GML feature, it is not recognized as a valid *city* feature. Note that in city.xsd the <Road> and <River> features are declared equivalent to ex:_CityFeature using the *substitutionGroup* attribute; this abstract element functions as a *label* that restricts membership in the <CityModel> feature collection--only features so labeled are allowable members, as defined by `CityMemberType`. This technique demonstrates the application of the "Feature Filter" (see 5.2.7) that restricts membership in GML feature collections.

One <cityMember> element in Listing 6.2 functions as a simple link by employing several XLink attributes; in effect we have a pointer entitled "Trinity Lane". Any <featureMember> element may behave as a simple link that references a remote resource. The link can point to a document fragment using an *xpointer scheme* that identifies a location, point, or range in the target document [XPointer]. In this case the value of the *href* attribute for the remote member contains an HTTP query string that can retrieve the feature instance; the *remoteSchema* attribute points to a schema fragment that constrains the instance: namely, the complex type definition in city.xsd that bears the name "RoadType".

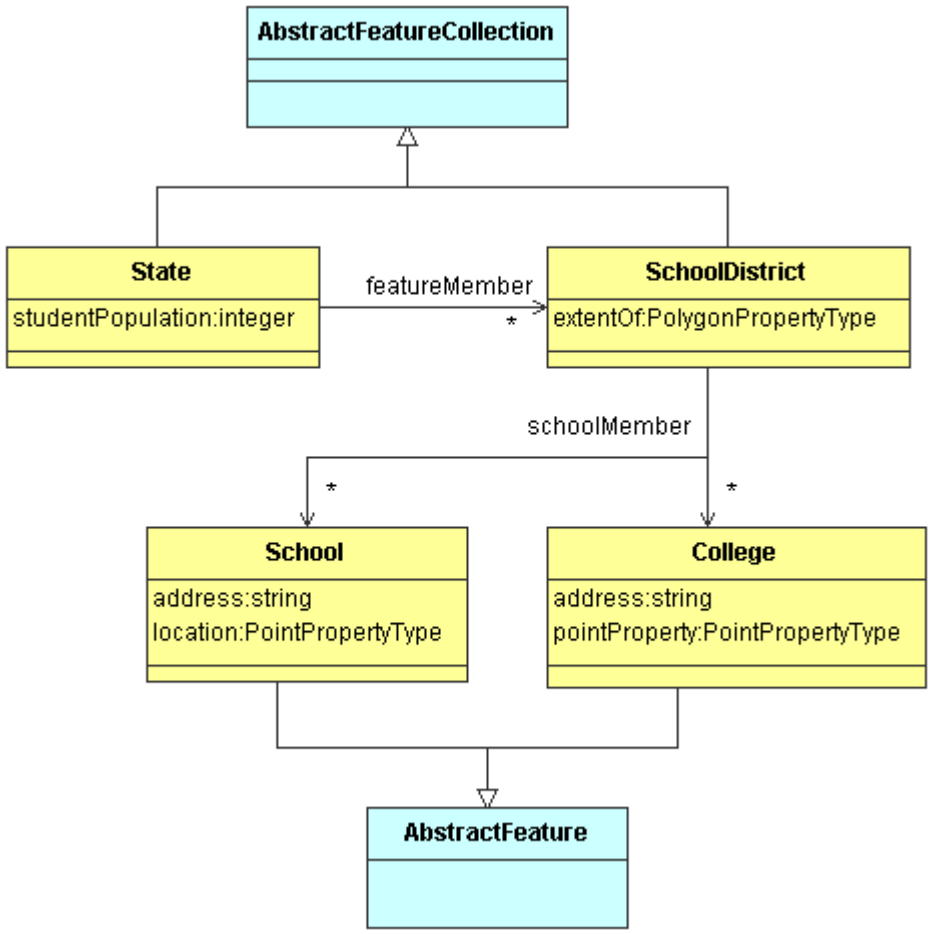
Listing 6.2: cambridge.xml View s

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- File: cambridge.xml -->
<CityModel xmlns="http://www.opengis.net/examples"
           xmlns:gml="http://www.opengis.net/gml"
           xmlns:xlink="http://www.w3.org/1999/xlink"
           xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
           xsi:schemaLocation="http://www.opengis.net/examples city.xsd">

  <gml:name>Cambridge</gml:name>
  <gml:boundedBy>
    <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <gml:coord><gml:X>0.0</gml:X><gml:Y>0.0</gml:Y></gml:coord>
      <gml:coord><gml:X>100.0</gml:X><gml:Y>100.0</gml:Y></gml:coord>
    </gml:Box>
  </gml:boundedBy>

  <cityMember>
    <River>
      <gml:description>The river that runs through Cambridge.</gml:descripti
      <gml:name>Cam</gml:name>
      <gml:centerLineOf>
        <gml:LineString srsName="http://www.opengis.net/gml/srs/epsg.xml#43
          <gml:coord><gml:X>0</gml:X><gml:Y>50</gml:Y></gml:coord>
          <gml:coord><gml:X>70</gml:X><gml:Y>60</gml:Y></gml:coord>
          <gml:coord><gml:X>100</gml:X><gml:Y>50</gml:Y></gml:coord>
        </gml:LineString>
      </gml:centerLineOf>
    </River>
  </cityMember>

  <cityMember>
    <Road>
      <gml:name>M11</gml:name>
        <linearGeometry>
          <gml:LineString srsName="http://www.opengis.net/gml/srs/epsg.xml#
            <gml:coord><gml:X>0</gml:X><gml:Y>5.0</gml:Y></gml:coord>
            <gml:coord><gml:X>20.6</gml:X><gml:Y>10.7</gml:Y></gml:coord>
            <gml:coord><gml:X>80.5</gml:X><gml:Y>60.9</gml:Y></gml:coord>
          </gml:LineString>
        </linearGeometry>
      <classification>motorway</classification>
      <number>11</number>
    </Road>
```

```
  </cityMember>

  <cityMember xlink:type="simple" xlink:title="Trinity Lane"
    xlink:href="http://www.foo.net/cgi-bin/wfs?FeatureID=C10239"
    gml:remoteSchema="city.xsd#xpointer(//complexType[@name='RoadType'])"/>

  <!-- a mountain doesn't belong here! Uncomment this cityMember and see
       the parser complain!
  <cityMember>
    <Mountain>
      <gml:description>World's highest mountain is in Nepal!</gml:descripti
      <gml:name>Everest</gml:name>
      <elevation>8850</elevation>
    </Mountain>
  </cityMember>
  -->

  <dateCreated>2000-11</dateCreated>
</CityModel>
```

Figure 6.2 is a UML diagram for the Schools example. The `SchoolDistrict`class is associated with the `State` class via the *featureMember* relationship, and instances of the `School`or `College` classes are members of the SchoolDistrict collection.

Figure 6.2: UML diagram for the Schools example



Listing 6.3 is an application schema for the Schools example. The purpose of this example is to demonstrate that feature collections may indeed contain other feature collections. To keep things fairly simple no attempt has been made to restrict membership in any of the collections; this means that a valid instance document could contain **any** GML feature within the <State> and <SchoolDistrict> collections, not just those pertaining to educational institutions. Sub-section 5.2.7 describes a design pattern for restricting collection membership.

Listing 6.3: schools.xsd **View so**

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- File: schools.xsd  -->
<schema targetNamespace="http://www.opengis.net/examples"
        xmlns:ex="http://www.opengis.net/examples"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xmlns:gml="http://www.opengis.net/gml"
        xmlns="http://www.w3.org/2000/10/XMLSchema"
        elementFormDefault="qualified" version="2.01">

  <annotation>
    <appinfo>schools.xsd v2.01 2001-02</appinfo>
    <documentation xml:lang="en">
```

```
    GML schema for Schools example.
  </documentation>
</annotation>

<!-- import constructs from the GML Feature and Geometry schemas -->
<import namespace="http://www.opengis.net/gml" schemaLocation="feature.xs

<!-- ======================================================================
     global element declarations
======================================================================= -->

<element name="State" type="ex:StateType"
  substitutionGroup="gml:_FeatureCollection"/>
<element name="SchoolDistrict" type="ex:SchoolDistrictType"
  substitutionGroup="gml:_FeatureCollection"/>
<element name="schoolMember" type="gml:FeatureAssociationType"
  substitutionGroup="gml:featureMember"/>
<element name="School" type="ex:SchoolType"
  substitutionGroup="gml:_Feature"/>
<element name="College" type="ex:CollegeType"
  substitutionGroup="gml:_Feature"/>
<element name="address" type="string"/>

<!-- ======================================================================
     type definitions for state educational institutions
======================================================================= -->

<complexType name="StateType">
  <complexContent>
    <extension base="gml:AbstractFeatureCollectionType">
      <sequence>
        <element name="studentPopulation" type="integer"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="SchoolDistrictType">
  <complexContent>
    <extension base="gml:AbstractFeatureCollectionType">
      <sequence>
        <element ref="gml:extentOf"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="SchoolType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element ref="ex:address"/>
        <element ref="gml:location"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

```
  <complexType name="CollegeType">
    <complexContent>
      <extension base="gml:AbstractFeatureType">
        <sequence>
          <element ref="ex:address"/>
          <element ref="gml:pointProperty" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</schema>
```

A few interesting things are happening in this example. The root <State> element is an instance of `ex:StateType`, which is derived from the abstract `gml:AbstractFeatureCollectionType` defined in the GML Feature schema. One of the child elements, <SchoolDistrict>, is also a feature collection; in effect we have a feature collection containing a feature collection as one of its members. Listing 6.4 is a conforming instance document. Refer to Box 1.2 for a summary of the example.

Listing 6.4: schools.xml Vie

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- File: schools.xml -->
<State xmlns="http://www.opengis.net/examples"
       xmlns:gml="http://www.opengis.net/gml"
       xmlns:xlink="http://www.w3.org/1999/xlink"
       xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
       xsi:schemaLocation="http://www.opengis.net/examples schools.xsd">

  <gml:description>
    Educational institutions with student populations exceeding 500.
  </gml:description>
  <gml:name>School districts in the North Region.</gml:name>
  <gml:boundedBy>
    <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <gml:coord><gml:X>0</gml:X><gml:Y>0</gml:Y></gml:coord>
      <gml:coord><gml:X>50</gml:X><gml:Y>50</gml:Y></gml:coord>
    </gml:Box>
  </gml:boundedBy>

  <gml:featureMember>
    <SchoolDistrict>
      <gml:name>District 28</gml:name>
      <gml:boundedBy>
        <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
          <gml:coord><gml:X>0</gml:X><gml:Y>0</gml:Y></gml:coord>
          <gml:coord><gml:X>50</gml:X><gml:Y>40</gml:Y></gml:coord>
        </gml:Box>
      </gml:boundedBy>

      <schoolMember>
        <School>
```

```
        <gml:name>Alpha</gml:name>
        <address>100 Cypress Ave.</address>
        <gml:location>
          <gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#432
            <gml:coord><gml:X>20.0</gml:X><gml:Y>5.0</gml:Y></gml:coord>
          </gml:Point>
        </gml:location>
      </School>
    </schoolMember>

    <schoolMember>
      <School>
        <gml:name>Beta</gml:name>
        <address>1673 Balsam St.</address>
        <gml:location>
          <gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#432
            <gml:coord><gml:X>40.0</gml:X><gml:Y>5.0</gml:Y></gml:coord>
          </gml:Point>
        </gml:location>
      </School>
    </schoolMember>

    <gml:extentOf>
      <gml:Polygon srsName="http://www.opengis.net/gml/srs/epsg.xml#4326":
        <gml:outerBoundaryIs>
          <gml:LinearRing>
            <gml:coord><gml:X>0</gml:X><gml:Y>0</gml:Y></gml:coord>
            <gml:coord><gml:X>50</gml:X><gml:Y>0</gml:Y></gml:coord>
            <gml:coord><gml:X>50</gml:X><gml:Y>40</gml:Y></gml:coord>
            <gml:coord><gml:X>0</gml:X><gml:Y>0</gml:Y></gml:coord>
          </gml:LinearRing>
        </gml:outerBoundaryIs>
      </gml:Polygon>
    </gml:extentOf>
  </SchoolDistrict>
</gml:featureMember>

<gml:featureMember>
  <SchoolDistrict>
    <gml:name>District 32</gml:name>
    <gml:boundedBy>
      <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
        <gml:coord><gml:X>0</gml:X><gml:Y>0</gml:Y></gml:coord>
        <gml:coord><gml:X>30</gml:X><gml:Y>50</gml:Y></gml:coord>
      </gml:Box>
    </gml:boundedBy>

    <schoolMember>
      <School>
        <gml:name>Gamma</gml:name>
        <address>651 Sequoia Ave.</address>
        <gml:location>
          <gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#432
            <gml:coord><gml:X>5.0</gml:X><gml:Y>20.0</gml:Y></gml:coord>
          </gml:Point>
        </gml:location>
      </School>
```

```
      </schoolMember>

      <schoolMember xlink:type="simple" xlink:href="http:abc.com">
        <College>
          <gml:name>Delta</gml:name>
          <address>260 University Blvd.</address>
          <gml:pointProperty>
            <gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#432
              <gml:coord><gml:X>5.0</gml:X><gml:Y>40.0</gml:Y></gml:coord>
            </gml:Point>
          </gml:pointProperty>
        </College>
      </schoolMember>

      <schoolMember xlink:type="simple" xlink:title="Epsilon High School"
        xlink:href="http:www.state.gov/schools/cgi-bin/wfs?schoolID=hs736"
        gml:remoteSchema="schools.xsd#xpointer(//complexType[@name='SchoolT

      <gml:extentOf>
        <gml:Polygon srsName="http://www.opengis.net/gml/srs/epsg.xml#4326":
          <gml:outerBoundaryIs>
            <gml:LinearRing>
              <gml:coord><gml:X>0</gml:X><gml:Y>0</gml:Y></gml:coord>
              <gml:coord><gml:X>40</gml:X><gml:Y>50</gml:Y></gml:coord>
              <gml:coord><gml:X>50</gml:X><gml:Y>50</gml:Y></gml:coord>
              <gml:coord><gml:X>0</gml:X><gml:Y>0</gml:Y></gml:coord>
            </gml:LinearRing>
          </gml:outerBoundaryIs>
        </gml:Polygon>
      </gml:extentOf>
    </SchoolDistrict>
  </gml:featureMember>
  <studentPopulation>392620</studentPopulation>
</State>
```

Note the use of <coord> elements to convey coordinate values; the XML parser constrains the number of tuples according to geometry type. For example, a <Point> element has exactly one coordinate tuple, and a <LinearRing> has at least four.

The final example illustrates the construction of a "horizontal" application schema in that it might be applied to a range of application domains. This example is simple data interchange schema that facilitates the exchange of application-level data structures (i.e. features and/or feature collections); such a schema provides a generic means of transferring instances of simple features.

Listing 6.5 is a sample instance document that conforms to the schema in Listing 6.6. A feature may include any of the predefined simple geometric properties (e.g. those that return Points, Polygons, or LineStrings). Non-spatial properties must reflect one of the atomic datatypes of XML Schema.

Listing 6.5: gmlpacket.xml **View source**

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- File: gmlpacket.xml -->
<gmlPacket xmlns="http://www.opengis.net/examples/packet"
           xmlns:gml="http://www.opengis.net/gml"
           xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
           xsi:schemaLocation=
             "http://www.opengis.net/examples/packet  gmlpacket.xsd">

  <gml:description>Road network elements</gml:description>
  <gml:boundedBy>
    <gml:null>unknown</gml:null>
  </gml:boundedBy>

  <packetMember>
    <StaticFeature fid="Highway-99" featureType="Road">
      <gml:centerLineOf>
        <gml:LineString>
          <gml:coordinates>...</gml:coordinates>
        </gml:LineString>
      </gml:centerLineOf>
      <property>
        <propertyName>numLanes</propertyName>
        <value dataType="integer">2</value>
      </property>
      <property>
        <propertyName>surfaceType</propertyName>
        <value dataType="string">asphalt</value>
      </property>
    </StaticFeature>
  </packetMember>

  <Metadata>
    <title>Vancouver-Squamish corridor</title>
  </Metadata>
</gmlPacket>
```

Listing 6.6 is the 'gmlpacket' schema. The <gmlPacket> element is the root feature collection; this schema restricts allowable feature members to instances of `pak:StaticFeatureType`. None of the type definitions in the gmlpacket schema can be extended or restricted in any manner, and this schema cannot serve as the basis for any other application schema (i.e. it cannot be imported or included into another schema).

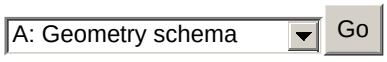
Listing 6.6: gmlpacket.xsd **View so**

```
<?xml version="1.0" encoding="UTF-8"?>
<!--  File: gmlpacket.xsd  -->
<schema targetNamespace="http://www.opengis.net/examples/packet"
        xmlns:pak="http://www.opengis.net/examples/packet"
        xmlns:gml="http://www.opengis.net/gml"
        xmlns="http://www.w3.org/2000/10/XMLSchema"
        elementFormDefault="qualified"
        finalDefault="#all" version="0.5">
```

```
<annotation>
  <appinfo>gmlpacket.xsd v0.5 2001-02</appinfo>
  <documentation xml:lang="en">
    GML schema for simple data transfer.
  </documentation>
</annotation>

<!-- import constructs from the GML Feature and Geometry schemas -->
<import namespace="http://www.opengis.net/gml" schemaLocation="feature.xs

<element name="gmlPacket" type="pak:GMLPacketType"/>
<element name="packetMember" type="pak:packetMemberType"/>
<element name="StaticFeature" type="pak:StaticFeatureType"/>

<complexType name="GMLPacketBaseType">
  <complexContent>
    <restriction base="gml:AbstractFeatureCollectionType">
      <sequence>
        <element ref="gml:description" minOccurs="0"/>
        <element ref="gml:name" minOccurs="0"/>
        <element ref="gml:boundedBy"/>
        <element ref="pak:packetMember" minOccurs="0" maxOccurs="unbounde
      </sequence>
      <attribute name="fid" type="ID" use="optional"/>
    </restriction>
  </complexContent>
</complexType>

<complexType name="GMLPacketType">
  <complexContent>
    <extension base="pak:GMLPacketBaseType">
      <sequence>
        <element name="Metadata" type="pak:MetadataType" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="packetMemberType">
  <complexContent>
    <restriction base="gml:FeatureAssociationType">
      <sequence minOccurs="0">
        <element ref="pak:StaticFeature"/>
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>

<complexType name="MetadataType">
  <sequence>
    <element name="origin" type="string" minOccurs="0"/>
    <element name="title" type="string" minOccurs="0"/>
    <element name="abstract" type="string" minOccurs="0"/>
  </sequence>
</complexType>
```

```
  <complexType name="StaticFeatureType">
    <complexContent>
      <extension base="gml:AbstractFeatureType">
        <sequence>
          <element ref="gml:pointProperty" minOccurs="0"/>
          <element ref="gml:polygonProperty" minOccurs="0"/>
          <element ref="gml:lineStringProperty" minOccurs="0"/>
          <element name="property" type="pak:PropertyType"
            minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="featureType" type="string" use="required"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="PropertyType">
    <sequence>
      <element name="propertyName" type="string"/>
      <element name="value">
        <complexType>
          <simpleContent>
            <extension base="string">
              <attribute name="dataType" type="pak:DataType"
                use="required"/>
            </extension>
          </simpleContent>
        </complexType>
      </element>
    </sequence>
  </complexType>

  <simpleType name="DataType">
    <restriction base="string">
      <enumeration value="string"/>
      <enumeration value="integer"/>
      <enumeration value="long"/>
      <enumeration value="decimal"/>
      <enumeration value="boolean"/>
      <enumeration value="time"/>
      <enumeration value="date"/>
    </restriction>
  </simpleType>
</schema>
```

7 Profiles of GML



GML is a fairly complex specification that is rich in functionality. In general an implementation need not exploit the entire specification, but may employ a subset of constructs corresponding to specific relevant requirements. A profile of GML could be defined to enhance interoperability and to curtail ambiguity by allowing only a specific subset of

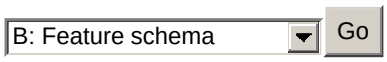
GML; different application schemas could conform to such a profile in order to take advantage of any interoperability or performance advantages that it offers in comparison with full-blown GML. Such profiles could be defined inside other OGC specifications.

There may be cases where reduced functionality is acceptable, or where processing requirements compel use of a logical subset of GML. For example, applications that don't need to handle XLink attributes in any form can adhere to a specific profile that excludes them; the constraint in this case would be to not use links. Other cases might include defining constraints on the level of nesting allowed inside tags (i.e. tree depth), or only allowing features with homogeneous properties as members of a feature collection. In many cases such constraints can be enforced via new schemas; others may be enforced through procedural agreements within an information community.

Here are the guidelines for developing a profile:

1. A profile of GML is a restriction of the basic descriptive capability of GML.
2. Any such profile must be fully compliant with the GML 2.0 specification.
3. GML abstract type definitions may be freely extended or restricted.
4. A profile must not change the name, definition, or data type of mandatory GML elements.
5. The schema or schemas that define a profile must be made available to any application schemas which will conform to the profile.
6. The relevant schema or schemas that define a profile must reside in a specified namespace that *must not* be `http://www.opengis.net/gml` (i.e. the core 'gml' namespace)

Appendix A: The Geometry schema, v2.06 (normative)




```
<?xml version="1.0" encoding="UTF-8"?>
<!--  File: geometry.xsd  -->
<schema targetNamespace="http://www.opengis.net/gml"
        xmlns="http://www.w3.org/2000/10/XMLSchema"
        xmlns:gml="http://www.opengis.net/gml"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        elementFormDefault="qualified"
        version="2.06">

  <annotation>
    <appinfo>geometry.xsd v2.06 2001-02</appinfo>
    <documentation xml:lang="en">
      GML Geometry schema. Copyright (c) 2001 OGC, All Rights Reserved.
    </documentation>
  </annotation>

  <!-- bring in the XLink attributes -->
```

```
<import namespace="http://www.w3.org/1999/xlink" schemaLocation="xlinks.xs

<!-- ==============================================================
     global declarations
================================================================== -->

<element name="_Geometry" type="gml:AbstractGeometryType" abstract="true"/
<element name="_GeometryCollection" type="gml:GeometryCollectionType"
  abstract="true"/>
<element name="geometryMember" type="gml:GeometryAssociationType"/>

<!-- primitive geometry elements -->
<element name="Point" type="gml:PointType" substitutionGroup="gml:_Geometr
<element name="LineString" type="gml:LineStringType"
  substitutionGroup="gml:_Geometry"/>
<element name="LinearRing" type="gml:LinearRingType"
  substitutionGroup="gml:_Geometry"/>
<element name="Polygon" type="gml:PolygonType"
  substitutionGroup="gml:_Geometry"/>
<element name="Box" type="gml:BoxType"/>

<!-- aggregate geometry elements -->
<element name="MultiGeometry" type="gml:GeometryCollectionType"/>
<element name="MultiPoint" type="gml:MultiPointType"
  substitutionGroup="gml:_Geometry"/>
<element name="MultiLineString" type="gml:MultiLineStringType"
  substitutionGroup="gml:_Geometry"/>
<element name="MultiPolygon" type="gml:MultiPolygonType"
  substitutionGroup="gml:_Geometry"/>

<!-- coordinate elements -->
<element name="coord" type="gml:CoordType"/>
<element name="coordinates" type="gml:CoordinatesType"/>

<!-- this attribute gives the location where an element is defined -->
<attribute name="remoteSchema" type="uriReference" />

<!-- ==============================================================
     abstract supertypes
================================================================== -->

<complexType name="AbstractGeometryType" abstract="true">
  <annotation>
    <documentation>
      All geometry elements are derived from this abstract supertype;
      a geometry element may have an identifying attribute ('gid').
      It may be associated with a spatial reference system.
    </documentation>
  </annotation>
  <attribute name="gid" type="ID" use="optional"/>
  <attribute name="srsName" type="uriReference" use="optional"/>
</complexType>

<complexType name="AbstractGeometryCollectionBaseType" abstract="true">
  <annotation>
    <documentation>
      This abstract base type for geometry collections just makes the
      srsName attribute mandatory.
```

```
    </documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:AbstractGeometryType">
      <attribute name="gid" type="ID" use="optional"/>
      <attribute name="srsName" type="uriReference" use="required"/>
    </restriction>
  </complexContent>
</complexType>

<attributeGroup name="AssociationAttributeGroup">
  <annotation>
    <documentation>
      These attributes can be attached to any element, thus allowing it
      to act as a pointer. The 'remoteSchema' attribute allows an element
      that carries link attributes to indicate that the element is declare
      in a remote schema rather than by the schema that constrains the
      current document instance.
    </documentation>
  </annotation>
  <attributeGroup ref="xlink:simpleLink"/>
  <attribute ref="gml:remoteSchema" use="optional"/>
</attributeGroup>

<complexType name="GeometryAssociationType">
  <annotation>
    <documentation>
      An instance of this type (e.g. a geometryMember) can either
      enclose or point to a primitive geometry element. When serving
      as a simple link that references a remote geometry instance,
      the value of the gml:remoteSchema attribute can be used to
      locate a schema fragment that constrains the target instance.
    </documentation>
  </annotation>
  <sequence>
    <element ref="gml:_Geometry" minOccurs="0"/>
  </sequence>
  <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>

<!-- ==========================================================
     primitive geometry types
========================================================== -->

<complexType name="PointType">
  <annotation>
    <documentation>
      A Point is defined by a single coordinate tuple.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="gml:AbstractGeometryType">
      <sequence>
        <choice>
          <element ref="gml:coord"/>
          <element ref="gml:coordinates"/>
        </choice>
      </sequence>
```

```
    </extension>
  </complexContent>
</complexType>

<complexType name="LineStringType">
  <annotation>
    <documentation>
      A LineString is defined by two or more coordinate tuples, with
      linear interpolation between them.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="gml:AbstractGeometryType">
      <sequence>
        <choice>
          <element ref="gml:coord" minOccurs="2" maxOccurs="unbounded"/>
          <element ref="gml:coordinates"/>
        </choice>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="LinearRingType">
  <annotation>
    <documentation>
      A LinearRing is defined by four or more coordinate tuples, with
      linear interpolation between them; the first and last coordinates
      must be coincident.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="gml:AbstractGeometryType">
      <sequence>
        <choice>
          <element ref="gml:coord" minOccurs="4" maxOccurs="unbounded"/>
          <element ref="gml:coordinates"/>
        </choice>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="BoxType">
  <annotation>
    <documentation>
      The Box structure defines an extent using a pair of coordinate tuple
    </documentation>
  </annotation>
  <complexContent>
    <extension base="gml:AbstractGeometryType">
      <sequence>
        <choice>
          <element ref="gml:coord" minOccurs="2" maxOccurs="2"/>
          <element ref="gml:coordinates"/>
        </choice>
      </sequence>
    </extension>
```

```
  </complexContent>
</complexType>

<complexType name="PolygonType">
  <annotation>
    <documentation>
      A Polygon is defined by an outer boundary and zero or more inner
      boundaries which are in turn defined by LinearRings.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="gml:AbstractGeometryType">
      <sequence>
        <element name="outerBoundaryIs">
          <complexType>
            <sequence>
              <element ref="gml:LinearRing"/>
            </sequence>
          </complexType>
        </element>
        <element name="innerBoundaryIs" minOccurs="0" maxOccurs="unbounded
          <complexType>
            <sequence>
              <element ref="gml:LinearRing"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<!-- ==============================================================
     aggregate geometry types
=================================================================== -->

<complexType name="GeometryCollectionType">
  <annotation>
    <documentation>
      A geometry collection must include one or more geometries, reference
      through geometryMember elements. User-defined geometry collections
      that accept GML geometry classes as members must instantiate--or
      derive from--this type.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="gml:AbstractGeometryCollectionBaseType">
      <sequence>
        <element ref="gml:geometryMember" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="MultiPointType">
  <annotation>
    <documentation>
      A MultiPoint is defined by one or more Points, referenced through
```

```
      pointMember elements.
    </documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:GeometryCollectionType">
      <sequence>
        <element name="pointMember" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element ref="gml:Point"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </restriction>
  </complexContent>
</complexType>

<complexType name="MultiLineStringType">
  <annotation>
    <documentation>
      A MultiLineString is defined by one or more LineStrings, referenced
      through lineStringMember elements.
    </documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:GeometryCollectionType">
      <sequence>
        <element name="lineStringMember" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element ref="gml:LineString"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </restriction>
  </complexContent>
</complexType>

<complexType name="MultiPolygonType">
  <annotation>
    <documentation>
      A MultiPolygon is defined by one or more Polygons, referenced throug
      polygonMember elements.
    </documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:GeometryCollectionType">
      <sequence>
        <element name="polygonMember" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element ref="gml:Polygon"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
```

```
      </restriction>
    </complexContent>
  </complexType>

  <!-- ====================================================================
       There are two ways to represent coordinates: (1) as a sequence
       of <coord> elements that encapsulate tuples, or (2) using a
       single <coordinates> string.
  ======================================================================= -->

  <complexType name="CoordType">
    <annotation>
      <documentation>
        Represents a coordinate tuple in one, two, or three dimensions.
      </documentation>
    </annotation>
    <sequence>
      <element name="X" type="decimal"/>
      <element name="Y" type="decimal" minOccurs="0"/>
      <element name="Z" type="decimal" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="CoordinatesType">
    <annotation>
      <documentation>
        Coordinates can be included in a single string, but there is no
        facility for validating string content. The value of the 'cs' attrib
        is the separator for coordinate values, and the value of the 'ts'
        attribute gives the tuple separator (a single space by default); the
        default values may be changed to reflect local usage.
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string">
        <attribute name="decimal" type="string" use="default" value="."/>
        <attribute name="cs" type="string" use="default" value=","/>
        <attribute name="ts" type="string" use="default" value="&#x20;"/>
      </extension>
    </simpleContent>
  </complexType>
</schema>
```

Appendix B: The Feature schema , v2.06 (normative)

C: XLinks schema Go

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- File: feature.xsd  -->
<schema targetNamespace="http://www.opengis.net/gml"
        xmlns:gml="http://www.opengis.net/gml"
        xmlns="http://www.w3.org/2000/10/XMLSchema"
        elementFormDefault="qualified"
        version="2.06">
```

```
<annotation>
  <appinfo>feature.xsd v2.06 2001-02</appinfo>
  <documentation xml:lang="en">
    GML Feature schema. Copyright (c) 2001 OGC, All Rights Reserved.
  </documentation>
</annotation>

<!-- include constructs from the GML Geometry schema -->
<include schemaLocation="geometry.xsd"/>

<!-- ==========================================================
     global declarations
========================================================== -->

<element name="_Feature" type="gml:AbstractFeatureType" abstract="true"/>
<element name="_FeatureCollection" type="gml:AbstractFeatureCollectionType
  abstract="true" substitutionGroup="gml:_Feature"/>
<element name="featureMember" type="gml:FeatureAssociationType"/>

<!-- some basic geometric properties of features -->
<element name="_geometryProperty" type="gml:GeometryPropertyType" abstract
<element name="geometryProperty" type="gml:GeometryPropertyType" />
<element name="boundedBy" type="gml:BoundingShapeType"/>

<element name="pointProperty" type="gml:PointPropertyType"
  substitutionGroup="gml:_geometryProperty"/>
<element name="polygonProperty" type="gml:PolygonPropertyType"
  substitutionGroup="gml:_geometryProperty"/>
<element name="lineStringProperty" type="gml:LineStringPropertyType"
  substitutionGroup="gml:_geometryProperty"/>
<element name="multiPointProperty" type="gml:MultiPointPropertyType"
  substitutionGroup="gml:_geometryProperty"/>
<element name="multiLineStringProperty" type="gml:MultiLineStringPropertyT
  substitutionGroup="gml:_geometryProperty"/>
<element name="multiPolygonProperty" type="gml:MultiPolygonPropertyType"
  substitutionGroup="gml:_geometryProperty"/>
<element name="multiGeometryProperty" type="gml:MultiGeometryPropertyType"
  substitutionGroup="gml:_geometryProperty"/>

<!-- common aliases for geometry properties -->
<element name="location" type="gml:PointPropertyType"
  substitutionGroup="gml:pointProperty"/>
<element name="centerOf" type="gml:PointPropertyType"
  substitutionGroup="gml:pointProperty"/>
<element name="position" type="gml:PointPropertyType"
  substitutionGroup="gml:pointProperty"/>
<element name="extentOf" type="gml:PolygonPropertyType"
  substitutionGroup="gml:polygonProperty"/>
<element name="coverage" type="gml:PolygonPropertyType"
  substitutionGroup="gml:polygonProperty"/>
<element name="edgeOf" type="gml:LineStringPropertyType"
  substitutionGroup="gml:lineStringProperty"/>
<element name="centerLineOf" type="gml:LineStringPropertyType"
  substitutionGroup="gml:lineStringProperty"/>
<element name="multiLocation" type="gml:MultiPointPropertyType"
  substitutionGroup="gml:multiPointProperty"/>
<element name="multiCenterOf" type="gml:MultiPointPropertyType"
```

```
  substitutionGroup="gml:multiPointProperty"/>
<element name="multiPosition" type="gml:MultiPointPropertyType"
  substitutionGroup="gml:multiPointProperty"/>
<element name="multiCenterLineOf" type="gml:MultiLineStringPropertyType"
  substitutionGroup="gml:multiLineStringProperty"/>
<element name="multiEdgeOf" type="gml:MultiLineStringPropertyType"
  substitutionGroup="gml:multiLineStringProperty"/>
<element name="multiCoverage" type="gml:MultiPolygonPropertyType"
  substitutionGroup="gml:multiPolygonProperty"/>
<element name="multiExtentOf" type="gml:MultiPolygonPropertyType"
  substitutionGroup="gml:multiPolygonProperty"/>

<!-- common feature descriptors -->
<element name="description" type="string"/>
<element name="name" type="string"/>

<!-- ================================================================
     abstract supertypes
================================================================== -->

<complexType name="AbstractFeatureType" abstract="true">
  <annotation>
    <documentation>
      An abstract feature provides a set of common properties. A concrete
      feature type must derive from this type and specify additional
      properties in an application schema. A feature may optionally
      possess an identifying attribute ('fid').
    </documentation>
  </annotation>
  <sequence>
    <element ref="gml:description" minOccurs="0"/>
    <element ref="gml:name" minOccurs="0"/>
    <element ref="gml:boundedBy" minOccurs="0"/>
    <!-- additional properties must be specified in an application schema
  </sequence>
  <attribute name="fid" type="ID" use="optional"/>
</complexType>

<complexType name="AbstractFeatureCollectionBaseType" abstract="true">
  <annotation>
    <documentation>
      This abstract base type just makes the boundedBy element mandatory
      for a feature collection.
    </documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:AbstractFeatureType">
      <sequence>
        <element ref="gml:description" minOccurs="0"/>
        <element ref="gml:name" minOccurs="0"/>
        <element ref="gml:boundedBy"/>
      </sequence>
      <attribute name="fid" type="ID" use="optional"/>
    </restriction>
  </complexContent>
</complexType>

<complexType name="AbstractFeatureCollectionType" abstract="true">
```

```
  <annotation>
    <documentation>
      A feature collection contains zero or more featureMember elements.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="gml:AbstractFeatureCollectionBaseType">
      <sequence>
        <element ref="gml:featureMember" minOccurs="0" maxOccurs="unbounde
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="GeometryPropertyType">
  <annotation>
    <documentation>
      A simple geometry property encapsulates a geometry element.
      Alternatively, it can function as a pointer (simple-type link)
      that refers to a remote geometry element.
    </documentation>
  </annotation>
  <sequence minOccurs="0">
    <element ref="gml:_Geometry"/>
  </sequence>
  <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>

<complexType name="FeatureAssociationType">
  <annotation>
    <documentation>
      An instance of this type (e.g. a featureMember) can either
      enclose or point to a feature (or feature collection); this
      type can be restricted in an application schema to allow only
      specified features as valid participants in the association.
      When serving as a simple link that references a remote feature
      instance, the value of the gml:remoteSchema attribute can be
      used to locate a schema fragment that constrains the target
      instance.
    </documentation>
  </annotation>
  <sequence minOccurs="0">
    <element ref="gml:_Feature"/>
  </sequence>
  <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>

<complexType name="BoundingShapeType">
  <annotation>
    <documentation>
      Bounding shapes--a Box or a null element are currently allowed.
    </documentation>
  </annotation>
  <sequence>
    <choice>
      <element ref="gml:Box"/>
      <element name="null" type="gml:NullType" />
    </choice>
```

```
  </sequence>
</complexType>

<!-- ================================================================
     geometry properties
================================================================== -->

<complexType name="PointPropertyType">
  <annotation>
    <documentation>
      Encapsulates a single point to represent position, location, or
      centerOf properties.
    </documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:GeometryPropertyType">
      <sequence minOccurs="0">
        <element ref="gml:Point"/>
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>

<complexType name="PolygonPropertyType">
  <annotation>
    <documentation>
      Encapsulates a single polygon to represent coverage or extentOf
      properties.
    </documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:GeometryPropertyType">
      <sequence minOccurs="0">
        <element ref="gml:Polygon"/>
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>

<complexType name="LineStringPropertyType">
  <annotation>
    <documentation>
      Encapsulates a single LineString to represent centerLineOf or
      edgeOf properties.
    </documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:GeometryPropertyType">
      <sequence minOccurs="0">
        <element ref="gml:LineString"/>
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>
```

```
<complexType name="MultiPointPropertyType">
  <annotation>
    <documentation>
      Encapsulates a MultiPoint element to represent the following
      discontiguous geometric properties: multiLocation, multiPosition,
      multiCenterOf.
    </documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:GeometryPropertyType">
      <sequence minOccurs="0">
        <element ref="gml:MultiPoint"/>
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>

<complexType name="MultiLineStringPropertyType">
  <annotation>
    <documentation>
      Encapsulates a MultiLineString element to represent the following
      discontiguous geometric properties: multiEdgeOf, multiCenterLineOf.
    </documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:GeometryPropertyType">
      <sequence minOccurs="0">
        <element ref="gml:MultiLineString"/>
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>

<complexType name="MultiPolygonPropertyType">
  <annotation>
    <documentation>
      Encapsulates a MultiPolygon to represent the following discontiguous
      geometric properties: multiCoverage, multiExtentOf.
    </documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:GeometryPropertyType">
      <sequence minOccurs="0">
        <element ref="gml:MultiPolygon"/>
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>

<complexType name="MultiGeometryPropertyType">
  <annotation>
    <documentation>Encapsulates a MultiGeometry element.</documentation>
  </annotation>
  <complexContent>
    <restriction base="gml:GeometryPropertyType">
```

```
        <sequence minOccurs="0">
          <element ref="gml:MultiGeometry"/>
        </sequence>
        <attributeGroup ref="gml:AssociationAttributeGroup"/>
      </restriction>
    </complexContent>
  </complexType>

  <simpleType name="NullType">
    <annotation>
      <documentation>
        If a bounding shape is not provided for a feature collection,
        explain why. Allowable values are:
        innapplicable - the features do not have geometry
        unknown - the boundingBox cannot be computed
        unavailable - there may be a boundingBox but it is not divulged
        missing - there are no features
      </documentation>
    </annotation>
    <restriction base="string">
      <enumeration value="inapplicable"/>
      <enumeration value="unknown"/>
      <enumeration value="unavailable"/>
      <enumeration value="missing"/>
    </restriction>
  </simpleType>
</schema>
```

Appendix C: The XLinks schema (normative)

At the time that GML 2.0 was finalised, the World Wide Web Consortium (W3C) had not produced a normative schema to support its XLink recommendation. As an interim measure, this schema has been produced by the editors of GML 2.0 to provide the XLink attributes for general use; pending the provision of a definitive schema by the W3C, this schema shall be considered a normative component of GML 2.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- File: xlinks.xsd  -->
<schema targetNamespace="http://www.w3.org/1999/xlink"
        xmlns="http://www.w3.org/2000/10/XMLSchema"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        version="2.01">

  <annotation>
    <appinfo>xlinks.xsd v2.01 2001-02</appinfo>
    <documentation xml:lang="en">
      This schema provides the XLink attributes for general use.
  </documentation>
  </annotation>

  <!-- ================================================================
       global declarations
  ================================================================= -->
```

```
<!-- locator attribute -->
<attribute name="href" type="uriReference" />

<!-- semantic attributes -->
<attribute name="role" type="uriReference" />
<attribute name="arcrole" type="uriReference" />
<attribute name="title" type="string" />

<!-- behavior attributes -->
<attribute name="show">
  <annotation>
    <documentation>
      The 'show' attribute is used to communicate the desired presentation
      of the ending resource on traversal from the starting resource; it's
      value should be treated as follows:
      new - load ending resource in a new window, frame, pane, or other
            presentation context
      replace - load the resource in the same window, frame, pane, or
                other presentation context
      embed - load ending resource in place of the presentation of the
              starting resource
      other - behavior is unconstrained; examine other markup in the
              link for hints
      none - behavior is unconstrained
    </documentation>
  </annotation>
  <simpleType>
    <restriction base="string">
      <enumeration value="new"/>
      <enumeration value="replace"/>
      <enumeration value="embed"/>
      <enumeration value="other"/>
      <enumeration value="none"/>
    </restriction>
  </simpleType>
</attribute>

<attribute name="actuate">
  <annotation>
    <documentation>
      The 'actuate' attribute is used to communicate the desired timing
      of traversal from the starting resource to the ending resource;
      it's value should be treated as follows:
      onLoad - traverse to the ending resource immediately on loading
               the starting resource
      onRequest - traverse from the starting resource to the ending
                  resource only on a post-loading event triggered for
                  this purpose
      other - behavior is unconstrained; examine other markup in link
              for hints
      none - behavior is unconstrained
    </documentation>
  </annotation>
  <simpleType>
    <restriction base="string">
      <enumeration value="onLoad"/>
      <enumeration value="onRequest"/>
```

```
      <enumeration value="other"/>
      <enumeration value="none"/>
    </restriction>
  </simpleType>
</attribute>

<!-- traversal attributes -->
<attribute name="label" type="string" />
<attribute name="from" type="string" />
<attribute name="to" type="string" />

<!-- ================================================================
     Attributes grouped by XLink type, as specified in the W3C
     Proposed Recommendation (dated 2000-12-20)
================================================================== -->

<attributeGroup name="simpleLink">
  <attribute name="type" type="string" use="fixed" value="simple"
    form="qualified"/>
  <attribute ref="xlink:href" use="optional"/>
  <attribute ref="xlink:role" use="optional"/>
  <attribute ref="xlink:arcrole" use="optional"/>
  <attribute ref="xlink:title" use="optional"/>
  <attribute ref="xlink:show" use="optional"/>
  <attribute ref="xlink:actuate" use="optional"/>
</attributeGroup>

<attributeGroup name="extendedLink">
  <attribute name="type" type="string" use="fixed" value="extended"
    form="qualified"/>
  <attribute ref="xlink:role" use="optional"/>
  <attribute ref="xlink:title" use="optional"/>
</attributeGroup>

<attributeGroup name="locatorLink">
  <attribute name="type" type="string" use="fixed" value="locator"
    form="qualified"/>
  <attribute ref="xlink:href" use="required"/>
  <attribute ref="xlink:role" use="optional"/>
  <attribute ref="xlink:title" use="optional"/>
  <attribute ref="xlink:label" use="optional"/>
</attributeGroup>

<attributeGroup name="arcLink">
  <attribute name="type" type="string" use="fixed" value="arc"
    form="qualified"/>
  <attribute ref="xlink:arcrole" use="optional"/>
  <attribute ref="xlink:title" use="optional"/>
  <attribute ref="xlink:show" use="optional"/>
  <attribute ref="xlink:actuate" use="optional"/>
  <attribute ref="xlink:from" use="optional"/>
  <attribute ref="xlink:to" use="optional"/>
</attributeGroup>

<attributeGroup name="resourceLink">
  <attribute name="type" type="string" use="fixed" value="resource"
    form="qualified"/>
  <attribute ref="xlink:role" use="optional"/>
```

```
    <attribute ref="xlink:title" use="optional"/>
    <attribute ref="xlink:label" use="optional"/>
  </attributeGroup>

  <attributeGroup name="titleLink">
    <attribute name="type" type="string" use="fixed" value="title"
      form="qualified"/>
  </attributeGroup>

  <attributeGroup name="emptyLink">
    <attribute name="type" type="string" use="fixed" value="none"
      form="qualified"/>
    </attributeGroup>
</schema>
```

Appendix D: References

[OGC00-0040]

Whiteside, A, and J. Bobbit. 2000. *Recommended Definition Data for Coordinate Reference Systems and Coordinate Transformations*. OGC Project Document 00-040r7.

[RFC2396]

Uniform Resource Identifiers (URI): Generic Syntax. (August 1998). Available [Online]: <ftp://www.ietf.org/rfc/rfc2396.txt>

[RFC2732]

Format for Literal IPv6 Addresses in URLs. (December 1999). Available [Online]: <http://www.ietf.org/rfc/rfc2732.txt>

[SVG]

Scalable Vector Graphics (SVG) 1.0 Specification. W3C Candidate Recommendation (2 November 2000). Available [Online]: <http://www.w3.org/TR/2000/CR-SVG-20001102/index.html>

[VRML200x]

The Virtual Reality Modeling Language. Draft International Standard ISO/IEC 14772:200x. Available [Online]: <http://www.web3d.org/TaskGroups/x3d/specification/>

[XLink]

XML Linking Language (XLink) Version 1.0. W3C Proposed Recommendation (20 December 2000). Available [Online]: <http://www.w3.org./TR/xlink/>

[XMLName]

Namespaces in XML. W3C Recommendation (14 January 1999). Available [Online]: <http://www.w3.org/TR/1999/REC-xml-names-19990114/>

[XMLSchema-1]

XML Schema Part 1: Structures. W3C Candidate Recommendation (24 October 2000). Available [Online]: <http://www.w3.org/TR/xmlschema-1/>

[XMLSchema-2]

XML Schema Part 2: Datatypes. W3C Candidate Recommendation (24 October 2000). Available [Online]: <http://www.w3.org/TR/xmlschema-2/>

[XPointer]

XML Pointer Language (XPointer) Version 1.0. W3C Candidate Recommendation (7 June 2000). Available [Online]: <http://www.w3.org./TR/xptr/>

Appendix E: Revision history

REC-GML2 (Recommendation, 2001-02-20):

- extensive restructuring of content
- fixed usage of 'type' attribute in xlinks utility schema
- the XLink schema is now normative
- feature collections can now be empty (per OGC Abstract Specification)
- renamed FeatureMemberType and GeometryMemberType to FeatureAssociationType and GeometryAssociationType, respectively
- updated UML diagrams
- added UML diagrams for the 'Cambridge' and 'Schools' examples
- expanded discussion about profiles and application frameworks (sec. 7)

PR-GML2 (Proposed Recommendation, 2001-01-15):

- Moved discussion of XLink attributes to section 4.4 and deleted the GeoLinks schema
- Removed material on temporal data (sec. 6) to a separate GML Module
- purged RDF remnants
- changed the names of identifying attributes ('fid' for features, 'gid' for geometry elements)
- added gml:AssociationType to support pointers to remote feature properties
- miscellaneous editorial changes

CR-GML2 (Candidate Recommendation, 2000-12-22):

- changed srsName attribute to type="uriReference"
- the featureMember element and all basic geometric properties can also function as pointers to remote resources (i.e. as simple-type XLink elements)
- added a brief introduction to section 5 that clarifies the different ways GML can be used to express associations (i.e. containment vs. linking)
- added an example to section 5.4 demonstrating the use of a linkbase to store instances of feature relationships (Listings 5.7 - 5.9)
- inserted comments concerning feature identification (sec. 4.2)
- miscellaneous corrections: invalid feature identifiers; invalid *xlink:role* attributes

- coordinate strings may now contain 1D, 2D, or 3D tuples
- added a description of the Feature Filter design pattern (sec. 4.4.6)
- removed the gml:label attribute from the GeoLinks schema