Annex E
(normative)

UML-to-GML application schema encoding rules

E.1 General concepts

The mapping from an ISO 19109 conformant UML Application Schema to the corresponding GML application schema is based on a set of encoding rules. These encoding rules are compliant with the rules for GML application schemas and are based on ISO 19118.

The rules are derived from the rules for the GML model and syntax as described in Clauses 7 to 21, especially Clause 7. The encoding rules of ISO 19118:2005, Annex A, are used whenever possible and feasible.

The rules listed in this annex aim at an automatic mapping from an ISO 19109 and ISO/TS 19103 conformant UML application schema to a GML application schema (in accordance with the rules defined in Clause 21). As a result of this automation, the resulting GML application schema will not make full use of the capabilities of XML and XML Schema, but will provide an XML implementation conformant to the ISO 19100 series of International Standards with a well-defined, predictable XML grammar.

These rules do not prescribe that all GML application schemas shall be generated by using these rules. All schemas following the rules defined in Clause 21 are valid and conformant GML application schemas, whether they are handcrafted, automatically derived from a UML application schema or produced by some other means.

The schema encoding rules are based on the general idea that the class definitions in the application schema are mapped to type and element declarations in XML Schema, so that the objects in the instance model can be mapped to corresponding element structures in the XML document.

E.2 Encoding rules

E.2.1 General encoding requirements

E.2.1.1 Application schemas

E.2.1.1.1 General (application schema, packages)

To be a valid input into the mapping the UML Application Schema shall conform to all of the following rules. See ISO 19118:2005, A.2.1, for additional requirements.

The UML Application Schema shall conform to the rules defined in ISO 19109 and ISO/TS 19103.

The UML Application Schema shall be represented by a package with the stereotype <<Application Schema>>. This package shall contain (i.e. own directly or indirectly) all UML model elements to be mapped to object types in the GML application schema. The package may include other packages without the stereotype <<Application Schema>> to group the different UML model elements within the application schema.

The UML model shall be complete and not contain external references unless exceptions are explicitly stated below. Predefined classes may be imported from the standardized schemas of the ISO 19100 series of

International Standards. The classes from the ISO 19100 series of International Standards that are implemented by the GML schema and used by the UML application schema shall be specified in a package with the name "ISO19100" or any sub-package of a package with that name.

Dependencies between packages shall be modelled explicitly. Permission elements with stereotype <<import>> or unspecified dependency elements between packages shall be used to express the dependency of elements in a package from elements in another package. All other dependency elements shall be ignored, see Figure E.1.

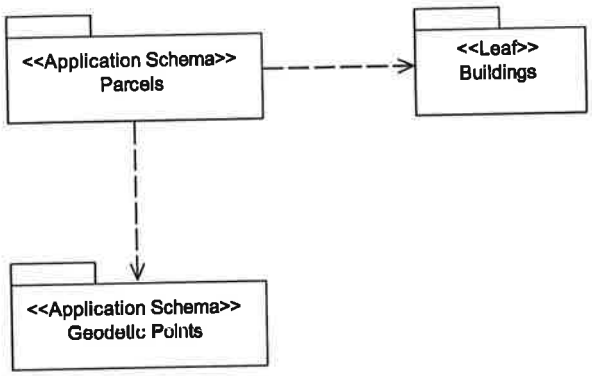

Figure E.1 — Dependency between packages <informative>

The visibility of all UML elements shall be set to "public". Only publicly visible elements shall be part of Application Schemas used for data interchange between applications.

Documentation of the elements in the UML model shall be stored in tagged values "documentation".

A unique XML namespace shall be associated with the UML Application Schema. Tagged values "targetNamespace" for the target namespace URI and "xmlns" for the abbreviation shall be set if and only if the package represents a UML application schema.

The version number of a package representing a UML Application Schema shall be specified in a tagged value "version", if applicable.

A GML profile may be associated with the application schema by a tagged value "gmlProfileSchema". If provided, the value shall be a URL referencing the schema location of the GML profile.

If a package shall be mapped to its own XML Schema document, a tagged value "xsdDocument" shall be set providing a valid relative file name of the schema document. The tagged value shall be set for every package representing the UML Application Schema. All tagged values "xsdDocument" in a UML model shall be unique.

EXAMPLE The value of an "xsdDocument" tagged value might be "GeodeticPoints.xsd" or "schemas/Parcels.xsd".

E.2.1.1.2 Classes

All class names within the same Application Schema shall be unique and an "NCName" as defined by W3C XML Namespaces:1999.

Feature types shall be modelled as UML classes with stereotype <<FeatureType>>, see Figure E.2.

NOTE 1 Neither ISO 19109 nor ISO 19118:2005, Annex A, distinguishes between feature types and object types — ISO 19109 only considers feature types while ISO 19118:2005, Annex A, classifies all feature types as object types. However,

the distinction is meaningful in GML and in practice often required in application schemas. The distinction made in this annex is a conformant refinement of ISO 19118:2005, Annex A.

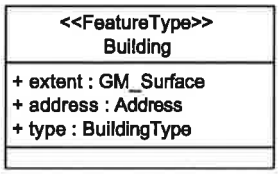

Figure E.2 — A feature type <informative>

Object types shall be modelled as UML classes with no stereotype. Object types are types where the instances shall have an identity, but which are not feature types[10].

EXAMPLE Examples of such types are geometries, topologies, reference systems. Instances of these types may have, for example, a name and an identifier.

UML classes with stereotype <<Type>> may have zero or more operations (these are not mapped to the GML application schema), attributes or associations.

The stereotype <<Abstract>> shall not be used in an Application Schema, because its use may be inconsistent with the use of correct UML notation, and thus misleading.

All instantiable subtypes of abstract types shall be either feature types, object types or data types.

Enumerations shall be modelled as UML classes with stereotype <<Enumeration>>.

Code lists shall be modelled as UML classes with stereotype <<CodeList>>, see Figure E.3.

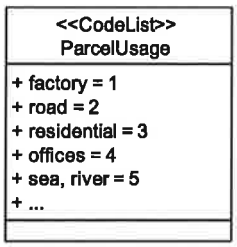

Figure E.3 — A code list <informative>

Union types shall be modelled as UML classes with stereotype <<Union>> (as specified in ISO 19107).

All other data types shall be modelled as UML classes with stereotype <<DataType>>, see Figure E.4.

10) Object types are not considered explicitly in ISO 19109:2005. They appear only as value types of property types.

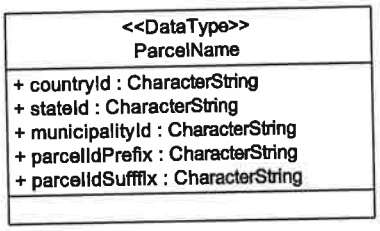

Figure E.4 — A data type <informative>

UML classes of the ISO 19100 series of International Standards that are part of the GML profile and for which a GML base type has been provided in Table D.2 in the "GML type" column may be subclassed in the UML application schema. In the subclasses, additional properties may be added or properties of the subtype may be redefined with a restricted multiplicity or domain of values.

NOTE 2 Although redefinition of properties is supported, these redefined properties will be ignored in the conversion rules and it is the responsibility of the application to verify the constraints introduced by the redefinition. All classes with other stereotypes than those mentioned above may be part of the UML Application Schema, but will be ignored.

NOTE 3 When an Application Schema refers to types defined by other standards of the ISO 19100 series which are implemented by the GML schema, then the class names should match one of those listed in the first column of Table D.2.

A generalization relationship may be specified only between two classes that are either:

— both feature types,

— both object types, or

— both data types.

All generalization relationships between classes shall have no stereotype. All generalization relationships with other stereotypes will be ignored. The discriminator property of the UML generalization shall be blank.

If a class is a specialization of another class, then this class shall have only one supertype (no support for multiple inheritance).

All classes shall have a stereotype specifying the meaning of the class. Classes without a stereotype are treated as object types, see Figure E.5.

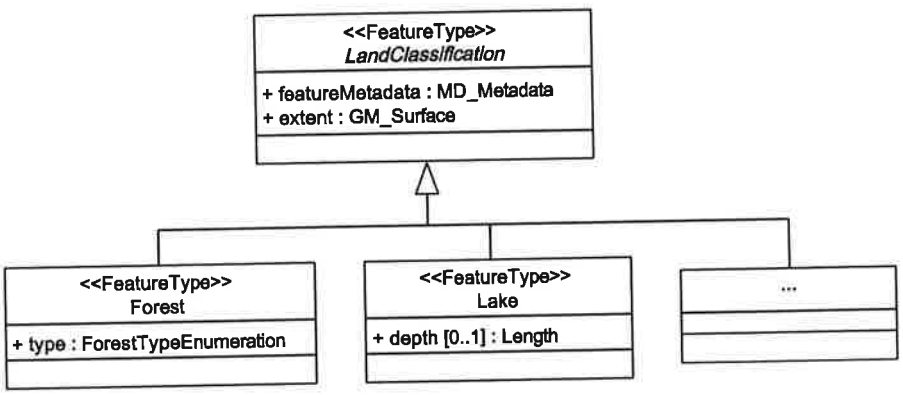

Figure E.5 — Generalization relationship between feature types <informative>

E.2.1.1.3 Attributes

Every UML attribute of an abstract type, feature type, object type, data type or union type shall have a name and a type. The name shall be an "NCName" as defined by W3C XML Namespaces:1999. If its multiplicity is not "1", the multiplicity shall be specified explicitly. An initial value may be specified for attributes with a number, string or enumeration type.

The type shall either be a predefined type (see E.2.1.1.5) or a class defined in the UML model.

Every UML attribute of an enumeration class shall have a name. The type information is left empty. No multiplicity, ordering or initial value information shall be attached to the attribute.

Every UML attribute of a code list class shall have a name. The type information is left empty. No multiplicity or ordering information shall be attached to the attribute. An initial value may be specified to document a code for the code list value. If it is omitted, the value (i.e. the attribute name) is used as the code.

The properties of a UML class are not ordered. To support the consistent ordering of the properties from the UML model in the conversion to XML Schema, a tagged value "sequenceNumber" (value domain: integer) shall be specified for every attribute. The value shall be unique for all attributes and association ends of a class.

E.2.1.1.4 Associations and association ends

Every UML association shall be an association with exactly two association ends. Both association ends shall connect to a feature, object or data type and shall have no stereotype or the stereotype <<association>> (otherwise the whole association will be ignored).

An association shall not contain any properties.

The rules for association ends are:

— If an association end is navigable it shall be marked as such and shall have a rolename. An association end with no name shall be ignored, even if it marked as navigable. If a name is provided, it shall be an "NCName" as defined by W3C XML Namespaces:1999.

— The multiplicity shall be given explicitly.

— The aggregation kind shall be specified explicitly if it is not "none".

— If the target class of an association end is a data type, then the aggregation kind shall be "composition".

Figure E.6 shows two example associations; one association is navigable in both directions and the other is an aggregation which is navigable in one direction only.

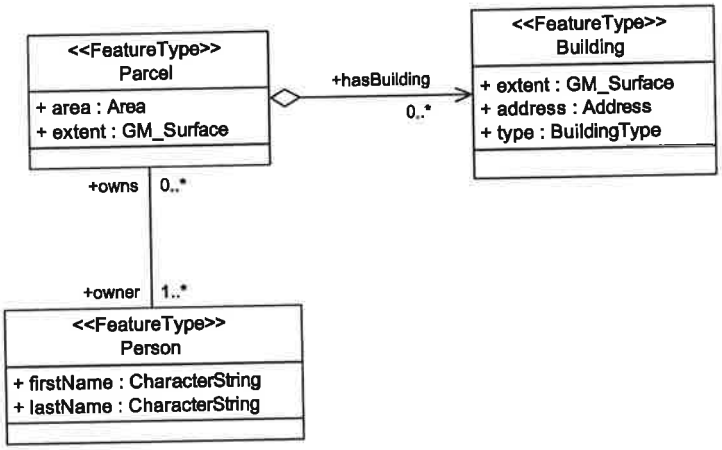

Figure E.6 — Associations <informative>

The properties of a UML class are not ordered. To support the consistent ordering of the properties from the UML model in the conversion to XML Schema, a tagged value "sequenceNumber" (value domain: integer) shall be specified for every association end. The value shall be unique for all attributes and association ends of a class.

E.2.1.1.5 Predefined types

The predefined types from ISO/TS 19103 listed in E.2.4.4 are treated as "basic types" in the sense of ISO 19118:2005, Annex A (i.e. a canonical XML Encoding is attached to them).

E.2.1.1.6 OCL constraints

All OCL constraints are ignored. The assessment of the validity of the instance model with respect to these constraints is the task of the application processing the GML instances.

NOTE The Schematron language may be used to express OCL constraints as part of the XML Schema representing the GML application schema.

E.2.1.1.7 Other information

All other information in the UML Application Schema is not used in the encoding rules and is ignored.

E.2.1.2 Character repertoire and languages

"UTF-8" or "UTF-16" shall be used as the character encoding of the XML Schema files (with the associated character repertoire) in accordance with XML.

E.2.1.3 Exchange metadata

Exchange metadata may be specified for every feature or feature collection in a GML instance document by specifying in the application schema property elements whose content model is derived from "`gml:AbstractMetadataPropertyType`" as described in E.2.4.11 and E.2.4.13.

No specific schema for the exchange metadata is added to the GML application schema.

E.2.1.4 Dataset and object identification

Unique identifiers in accordance with XML's ID mechanism are used to identify objects.

NOTE The XML ID mechanism only requires that these identifiers are unique identifiers within the XML document in which they appear.

E.2.1.5 Update mechanism

No explicit update mechanism is defined for the features defined in the GML application schema. It is assumed that other mechanisms are used to update a data store.

NOTE An example is the "Transaction" operation of the OpenGIS® Web Feature Service Implementation Specification.

E.2.2 Input data structure

See ISO 19118:2005, A.3, for a description of the input data structure.

E.2.3 Output data structure

This encoding rule is based on the XML Recommendation 1.0 and the XML Linking Language (XLink) Version 1.0. The schema for the output data structure that governs the structure of the exchange format shall be a (set of) valid XML Schema(s) in accordance with XML Schema 1.0 and the Rules for Application Schemas (see Clause 21).

The XML Schema conversion rules are defined in the following Subclause.

E.2.4 Conversion rules

E.2.4.1 General concepts

The schema conversion rules define how XML Schema documents (XSDs) shall be derived from an application schema expressed in UML in accordance with ISO 19109. A number of general rules are defined in E.2.4 to describe the mapping from a UML model that follows the guidelines described in E.2.1.

NOTE In this annex the namespace "xsd:" is used to refer to the namespace of XML Schema, which is "http://www.w3.org/2001/XMLSchema". The namespace "gml:" refers to the namespace of GML, which is "http://www.opengis.net/gml/3.2".

The rules are based on the GML model and syntax as described in Clauses 7 to 21 (especially Clauses 7, 9 and 21) and also on the encoding rules of ISO 19118:2005, Annex A.

The schema encoding rules are based on the general idea that the class definitions in the UML application schema are mapped to type and element declarations in XML Schema, so that the objects in the instance model can be mapped to corresponding element structures in the XML document.

Table E.1 gives an overview.

Table E.1 — Schema encoding overview

Table: UML → GML application schema overview	
UML application schema	**GML application schema**
Package	One XML Schema document per package (default mapping)
<<Application Schema>>	XML Schema document
<<DataType>>	Global element, whose content model is a globally scoped XML Schema complexType, property type
<<Enumeration>>	Restriction of xsd:string with enumeration values
<<CodeList>>	Union of an enumeration and a pattern (default mapping, an alternative mapping is a reference to a dictionary)
<<Union>>	Choice group whose members are GML objects or features, or objects corresponding to DataTypes
<<FeatureType>>	Global element, whose content model is a globally scoped XML Schema type derived by direct/indirect extension of gml:AbstractFeatureType, property type
No stereotype or <<Type>>	Global element, whose content model is a globally scoped XML Schema type derived by direct/indirect extension of gml:AbstractGMLType, property type
Operations	Not encoded
Attribute	local xsd:element, the type is either a property type (if the type is a complex type) or a simple type.
Association role	local xsd:element, the type is always a property type (only named and navigable roles)
General OCL constraints	Not encoded

NOTE <<FeatureType>> is a new stereotype which does not appear in ISO/TS 19103 or ISO 19109, and is used to indicate that the type is a realization of GF_FeatureType and a specialization from AbstractFeature.

The multiplicity of attributes and association roles is mapped to "minOccurs" and "maxOccurs" attributes in <xsd:element> declarations. The detailed mapping rules are described below.

For different UML model elements, different tagged values are used to control the mapping from UML to XML Schema. The following Table E.2 provides a list of these tagged values.

Table E.2 — Tagged values

UML model element	Associated tagged values
Package	— documentation — xsdDocument — targetNamespace (only <<Application Schema>>) — xmlns (only <<Application Schema>>) — version (only <<Application Schema>>) — gmlProfileSchema (only <<Application Schema>>)
Class	— documentation — noPropertyType — byValuePropertyType — isCollection — asDictionary (only <<CodeList>>) — xmlSchemaType (only <<Type>>)
Attribute and association end	— documentation — sequenceNumber — inlineOrByReference — isMetadata

E.2.4.2 UML packages

One XML Schema document is generated per package with the tagged value "xsdDocument" with the file name specified by the tagged value.

If the tagged value "xsdDocument" is set for a package, then the schema document contains all the XML Schema components resulting from the UML classes directly owned by the package. If the package is not a UML application schema, the schema document shall be included by the schema document that contains the schema components of the package that owns that package.

If the tagged value "xsdDocument" is not set for a package, all schema components are declared in the schema document that contains the schema components of the package that owns that package.

NOTE The tagged value is mandatory for all packages with the stereotype <<Application Schema>>, but optional for all other packages.

For every schema document, the "targetNamespace" and the "version" attributes of the root element shall be set in accordance with the tagged values of the same name in the package representing the UML Application Schema that owns the schema components within the schema document; if the "version" tagged value is not specified, the value "unknown" shall be used. In addition an "xmlns" attribute shall be specified for the target namespace with the value of the tagged value "xmlns" as the abbreviation.

EXAMPLE 1 "http://www.myorg.com/myns" may be a target namespace and "myns" may be the associated abbreviation used in the schema documents.

For every tagged value "gmlProfileSchema" of a package with the stereotype <<Application Schema>>, an element <gml:gmlProfileSchema> with the content of the tagged value shall be created in an appinfo annotation of the <schema> element as specified in 20.5.

The dependencies between the packages shall be used to determine the required imports of other schemas and additional includes of other schema documents:

— If the schema components specified by the target package of the dependency relationship are in the same target namespace as those of the supplier package, then the schema document specifying the schema components of the target package is "included".

— Otherwise the schema document representing the UML Application Schema package that contains the target package is "imported".

EXAMPLE 2 Mapping the information from Figure E.1 may result in:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.myorg.com/parcels" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:gp="http://www.myorg.com/geodeticPoints"
xmlns:pcl="http://www.myorg.com/parcels" xmlns:iso19115="http://www.isotc211.org/iso19115/"
xmlns:xlink="http://www.w3.org/1999/xlink" elementFormDefault="qualified" version="2003-07-20">
    <include schemaLocation="Buildings.xsd"/>
    <import namespace="http://www.myorg.com/geodeticPoints" schemaLocation="GeodeticPoints.xsd"/>
    <import namespace="http://www.opengis.net/gml/3.2" schemaLocation="base/gml.xsd"/>
    <!-- ... -->
</schema>
```

E.2.4.3 UML classes (general rules)

Recognized stereotypes for UML classes are: no stereotype, <<FeatureType>>, <<Type>>, <<DataType>>, <<Union>>, <<CodeList>>, and <<Enumeration>>. All classes will be mapped to the corresponding class category. All UML classes with other stereotypes will be ignored.

All UML classes shall have zero or one supertype.

All UML classes are mapped to named types. A suffix "Type" is added to the name of the type.

E.2.4.4 UML classes (basic types)

The basic types from the GML profile of ISO/TS 19103 listed in the left column of Table D.2 (starting with "CharacterString") are predefined and may be used as a data type of an attribute in an application schema conforming to ISO 19109. The mapping to a built-in type of XML Schema ("xsd:") or GML ("gml:") is specified. If multiple names are given in a cell of the table then the name in bold typeface shall be used as the default type of the mapping.

NOTE Multiple values in the right column are used to support also the reverse mapping in Annex F.

EXAMPLE ISO/TS 19103 Integer maps to "xsd:integer".

If a class with the stereotype <<Type>> has a canonical XML Schema encoding (e.g. from XML Schema) the XML Schema typename corresponding to the data type shall be given as the value of the tagged value "xmlSchemaType".

NOTE Canonical encodings may be preferred to structured encodings that follow the standard UML-to-GML encoding rules in some cases, for example where a compact structure based on "simpleContent" is already well known within the application domain.

E.2.4.5 UML classes (data types)

UML classes with stereotype <<DataType>> shall be mapped to XML Schema complex types.

NOTE Data types with other stereotypes, i.e. <<Enumeration>>, <<CodeList>> and <<Union>>, and predefined basic types are treated differently. See E.2.4.4, E.2.4.8, E.2.4.9, and E.2.4.10.

If the class has no supertype, it is a non-derived type in XML Schema; otherwise it extends its supertype which shall not be derived from `gml:AbstractGMLType` (directly or indirectly). Abstract superclasses without any attribute or navigable association role are ignored.

Global XML elements with appropriate settings for name (name of the UML class), type (name of the UML class plus "Type"), abstractness (if the class is abstract) and substitution groups (the qualified element name of the superclass or `gml:AbstractObject`, if the class has no superclass) shall be defined for these classes.

A named complex type shall be created for these classes (carrying the name of the class with a "PropertyType" suffix), if the class does not carry a tagged value "noPropertyType" with the value "true". The type follows the pattern for association properties as defined in GML (see 7.2.3), but without allowing Xlink attributes.

EXAMPLE The data type "ParcelName" from Figure E.4 may be mapped to:

```
<complexType name="ParcelNameType">
    <sequence>
        <element name="countryId" type="string"/>
        <element name="stateId" type="string"/>
        <element name="municipalityId" type="string"/>
        <element name="parcelIdPrefix" type="string"/>
        <element name="parcelIdSuffix" type="string" minOccurs="0"/>
    </sequence>
</complexType>

<element name="ParcelName" type="ex:ParcelNameType" substitutionGroup="gml:AbstractObject"/>
<complexType name="ParcelNamePropertyType">
    <sequence>
        <element ref="ex:ParcelName"/>
    </sequence>
    <attributeGroup ref="gml:OwnershipAttributeGroup" />
</complexType>
```

E.2.4.6 UML classes (feature types)

UML classes with stereotype <<FeatureType>> derive directly or indirectly from `gml:AbstractFeatureType`. If the class is a class without supertype, it extends directly `gml:AbstractFeatureType`; otherwise it extends its supertype which shall be derived from `gml:AbstractFeatureType` (again, directly or indirectly).

— Global XML elements with appropriate settings for name (name of the UML class), type (name of the UML class plus "Type"), abstractness (true, if the class is abstract) and substitution group (the name of the superclass or `gml:AbstractFeature`) are defined for these classes.

— If the class has a single association which is an aggregation or composition of a target class, the association role is converted to a property element, and the class carries a tagged value "isCollection" with the value "true", the attribute group `gml:AggregationAttributeGroup` is added to the complex type of the feature type.

— A named complex type shall be created for these classes (carrying the name of the class with a "PropertyType" suffix), if the class does not carry a tagged value "noPropertyType" with the value "true". The type follows the pattern for association properties as defined in GML (see 7.2.3).

— A named complex type shall be created for these classes (carrying the name of the class with a "PropertyByValueType" suffix), if the class carries a tagged value "byValuePropertyType" with the value "true". The type is a profile of the pattern for association properties as defined in GML restricted to the "by value" form (again, see 7.2.3).

EXAMPLE "Building" from Figure E.2 may be mapped to:

```
<complexType name="BuildingType">
    <complexContent>
        <extension base="gml:AbstractFeatureType">
            <sequence>
                <element name="extent" type="gml:SurfacePropertyType"/>
                <element name="address" type="pcl:AddressPropertyType"/>
                <element name="type" type="pcl:BuildingTypeType"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="BuildingPropertyType">
    <sequence minOccurs="0">
        <element ref="pcl:Building"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup" />
</complexType>

<complexType name="BuildingPropertyByValueType">
    <sequence>
        <element ref="pcl:Building"/>
    </sequence>
    <attributeGroup ref="gml:OwnershipAttributeGroup" />
</complexType>

<element name="Building" type="pcl:BuildingType" substitutionGroup="gml:AbstractFeature"/>
```

E.2.4.7 UML classes (object types)

UML classes with no stereotype or stereotype <<Type>> derive directly or indirectly from `gml:AbstractGMLType`. If the class is a class without supertype it extends directly `gml:AbstractGMLType`, otherwise it extends its supertype which shall be derived from `gml:AbstractGMLType` (again, directly or indirectly), but not from `gml:AbstractFeatureType` (again, directly or indirectly).

— Global XML elements with appropriate settings for name (name of the UML class), type (name of the UML class plus "Type"), abstractness (true, if the class is abstract) and substitution group (the name of the supertype or "AbstractGML") are defined for these classes.

— If the class has a single association which is an aggregation or composition of a target class, the association role is converted to a property element, and the class carries a tagged value "isCollection" with the value "true", the attribute group gml:AggregationAttributeGroup is added to the complex type of the object type.

— A named complex type shall be created for these classes (carrying the name of the class with a "PropertyType" suffix), if the class does not carry a tagged value "noPropertyType" with the value "true". The type follows the pattern for association properties as defined in GML (see 7.2.3).

— A named complex type shall be created for these classes (carrying the name of the class with a "PropertyByValueType" suffix), if the class carries a tagged value "byValuePropertyType" with the value "true".

The type is a profile of the pattern for association properties as defined in GML restricted to the "by value" form (again, see 7.2.3).

EXAMPLE

```
<element name="Ellipse" type="ex:EllipseType" substitutionGroup="gml:AbstractCurveSegment"/>

<complexType name="EllipseType">
    <complexContent>
        <extension base="gml:AbstractCurveSegmentType">
        <sequence>
                <element name="center" type="gml:DirectPositionType"/>
                <element name="semiminor" type="gml:VectorType"/>
                <element name="semimajor" type="gml:VectorType"/>
        </sequence>
        </extension>
    </complexContent>
</complexType>
```

E.2.4.8 UML classes (enumerations)

UML classes with stereotype <<Enumeration>> are mapped to XML Schema simple types. The base type is "string", the domain of values is restricted to the set of literal values as specified by the attribute names of the UML class.

EXAMPLE

```
    <simpleType name="SignType">
        <restriction base="string">
            <enumeration value="-"/>
            <enumeration value="+"/>
        </restriction>
    </simpleType>
```

E.2.4.9 UML classes (code lists)

A UML class with stereotype <<CodeList>> and without a tagged value "asDictionary" with the value "true" shall be mapped like an enumeration, but with the following differences:

— A facet "<pattern value='other: \w{2,}'/>" shall be added that allows for any text value beside the predefined values; these free values are prefixed with "other: ".

— If a code is specified for a code list value, only the code shall be represented as an enumeration facet.

— An encoded code value shall be qualified with an appinfo annotation with a `gml:description` element specifying the text value of the enumerated value.

EXAMPLE 1 The code list "ParcelUsage" from Figure E.3 may be represented as:

```
<simpleType name="ParcelUsageType">
    <union memberTypes="pcl:ParcelUsageEnumerationType pcl: ParcelUsageOtherType"/>
</simpleType>

<simpleType name="ParcelUsageEnumerationType">
    <restriction base="string">
        <enumeration value="1">
```

```
            <annotation>
                <appinfo><gml:description>factory</gml:description></appinfo>
            </annotation>
        </enumeration>
        <enumeration value="2">
            <annotation>
                <appinfo><gml:description>road</gml:description></appinfo>
            </annotation>
        </enumeration>
        <enumeration value="3">
            <annotation>
                <appinfo><gml:description>residential</gml:description></appinfo>
            </annotation>
        </enumeration>
        <enumeration value="4">
            <annotation>
                <appinfo><gml:description>offices</gml:description></appinfo>
            </annotation>
        </enumeration>
        <enumeration value="5">
            <annotation>
                <appinfo><gml:description>sea, river</gml:description></appinfo>
            </annotation>
        </enumeration>
    </restriction>
</simpleType>

<simpleType name="ParcelUsageOtherType">
    <restriction base="string">
        <pattern value="other: \w{2,}"/>
    </restriction>
</simpleType>
```

Alternatively, if the class carries a tagged value "asDictionary" with the value "true", a `gml:Dictionary` shall be used to represent a code list.

EXAMPLE 2 The code list "ParcelUsage" from Figure E.3 may be represented in a GML dictionary document as:

```
<gml:Dictionary gml:id="CodeList" xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/gml/3.2 gml.xsd">
    <gml:identifier codeSpace="http://www.someorg.de/cl.xml">My code lists</gml:identifier>
    <gml:dictionaryEntry>
        <gml:Dictionary gml:id="ParcelUsage">
            <gml:identifier codeSpace="http://www.someorg.de/cl.xml">ParcelUsage</gml:identifier>
            <gml:dictionaryEntry>
                <gml:Definition gml:id="ParcelUsage_1">
                    <gml:description>factory</gml:description>
                    <gml:identifier codeSpace="http://www.someorg.de/cl.xml#ParcelUsage">1</gml:identifier>
                </gml:Definition>
            </gml:dictionaryEntry>
            <gml:dictionaryEntry>
                <gml:Definition gml:id="ParcelUsage_2">
                    <gml:description>road</gml:description>
                    <gml:identifier codeSpace="http://www.someorg.de/cl.xml#ParcelUsage">2</gml:identifier>
                </gml:Definition>
            </gml:dictionaryEntry>
            <gml:dictionaryEntry>
                <gml:Definition gml:id="ParcelUsage_3">
                    <gml:description>residential</gml:description>
                    <gml:identifier codeSpace="http://www.someorg.de/cl.xml#ParcelUsage">3</gml:identifier>
                </gml:Definition>
```

```
            </gml:dictionaryEntry>
            <gml:dictionaryEntry>
                <gml:Definition gml:id="ParcelUsage_4">
                    <gml:description>offices</gml:description>
                    <gml:identifier codeSpace="http://www.someorg.de/cl.xml#ParcelUsage">4</gml:identifier>
                </gml:Definition>
            </gml:dictionaryEntry>
            <gml:dictionaryEntry>
                <gml:Definition gml:id="ParcelUsage_5">
                    <gml:description>sea, river</gml:description>
                    <gml:identifier codeSpace="http://www.someorg.de/cl.xml#ParcelUsage">5</gml:identifier>
                </gml:Definition>
            </gml:dictionaryEntry>
        </gml:Dictionary>
    </gml:dictionaryEntry>
</gml:Dictionary>
```

In an instance document the reference would then be encoded (using `gml:CodeType` as the content model, see E.2.4.11) for example as:

```
<usage codeSpace="http://www.someorg.de/example/cl.xml#ParcelUsage">1</usage>
```

The codeSpace attribute points to the dictionary, the value is the name of the entry in that dictionary.

The way a code list is encoded in a GML application schema also determines how property elements that carry the code lists as its value domain shall be encoded; see E.2.4.11.

E.2.4.10 UML classes (unions)

UML classes with stereotype <<Union>> are mapped as XML Schema complex types. These classes are mapped like data types (see E.2.4.5), but instead of a <xsd:sequence> of the properties, a <xsd:choice> is used so that exactly one of the properties is specified in an instance of a union.

EXAMPLE

```
<complexType name="RemoteResourceType">
    <choice>
        < element name="name" type="string"/>
        < element name="uri" type="anyURI"/>
    </choice>
</complexType>
```

E.2.4.11 UML attributes and association roles

A UML attribute or association role of an object or feature type is mapped to a local element with the same name in the complex type defining the content model of the object or feature type. The minOccurs and maxOccurs attributes are set in accordance with the definitions in the UML model (see ISO 19118:2005, Annex A, for details of the mapping). The type depends on the type of the value of the property in UML:

If the type of the value of the property is of simple content, then the type is used directly.

EXAMPLE 1 `<element name="count" type="integer"/>`

If the type of the value of the property is of complex content, then a property type shall be used. The default encoding of the property type allows both the inline or by-reference representation for feature and object types and the inline representation for data and union types. For feature and object types the representation may be restricted to inline or by-reference using a tagged value "inlineOrByReference" with the values "inline" or

"byReference" respectively. If the tagged value is missing or its value is "inlineOrByReference" the default encoding shall be used.

If an attribute or association role is a metadata property, then the property type shall extend `gml:AbstractMetadataPropertyType` (see 7.2.6); a metadata property is a property with the tagged value "isMetadata" with the value "true" or whose value is a class defined by ISO 19115:2003. If an association role is the target end of an aggregation or composition, then the property type shall extend `gml:AbstractMemberType` (see 7.2.5.1) unless it is a metadata property. If an association role is the target end of a composition or an object-valued attribute, then the property element shall add a Schematron constraint that asserts that the `owns` attribute of the `gml:OwnershipAttributeGroup` is "true". The Schematron constraint shall follow the following pattern:

```
<sch:pattern>
    <sch:rule context="qualified name of the object element">
        <sch:report test="qualified property name/@owns='true'">This property is a composition, values must be
owned</sch:report>
    </sch:rule>
</sch:pattern>
```

EXAMPLE 2 For a property ex:representativeLocation of a feature type ex:MyFeature that controls the point object describing the location this could be described as follows:

```
<element name="representativeLocation" type="gml:PointPropertyType">
    <annotation>
        <appinfo>
            <sch:pattern">
                <sch:rule context="ex:MyFeature">
                    <sch:report test="ex:representativeLocation/@owns='true'">This property is a composition, values
                    must be owned</sch:report>
                </sch:rule>
            </sch:pattern>
        </appinfo>
    </annotation>
</element>
```

If the property type is already specified in its application schema as a named type (this can be detected by inspecting the tagged values "noPropertyType" and "byValuePropertyType"), this schema component shall be referenced; otherwise, an anonymous property type shall be defined locally in the property element.

If the encoded property is an association end and the other association end of the association is also encoded in the GML application schema, the property name of the other association end shall be encoded in a `gml:reversePropertyName` element in an appinfo annotation of the property element (see 7.2.3.9).

EXAMPLE 3 By-reference or inline:

```
<element name="owner" type="ex:PersonPropertyType" minOccurs="0">
    <annotation>
        <appinfo>
                <gml:reversePropertyName>ex:owns</gml:reversePropertyName>
            </appinfo>
    </annotation>
</element>
...
<complexType name="PersonPropertyType">
    <sequence minOccurs="0">
        <element ref="ex:Person"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup" />
</complexType>
```

or

```
<element name="owner" minOccurs="0">
    <annotation>
        <appinfo>
            <gml:reversePropertyName>ex:owns</gml:reversePropertyName>
        </appinfo>
    </annotation>
    <complexType>
        <sequence minOccurs="0">
            <element ref="ex:Person"/>
        </sequence>
        <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </complexType>
</element>
```

Alternatively, the property type may support only one of the representations, inline or by-reference, depending on the tagged value "inlineOrByReference".

EXAMPLE 4 inline only:

```
<element name="owner" type="ex:PersonPropertyByValueType" minOccurs="0"/>
...
<complexType name="PersonPropertyByValueType">
    <sequence>
        <element ref="ex:Person"/>
    </sequence>
</complexType>
```

or

```
<element name="owner" minOccurs="0">
    <complexType>
        <sequence>
            <element ref="ex:Person"/>
        </sequence>
    </complexType>
</element>
```

If only the by-reference representation is to be supported, then the property element shall be qualified with an appinfo annotation element `gml:targetElement` specifying the qualified element name of the target type.

```
    <element name="targetElement" type="string"/>
```

If the encoded property is an association end and the other association end of the association is also encoded in the GML application schema, the property name of the other association end shall be encoded in another appinfo annotation element `gml:reversePropertyName` specified above.

EXAMPLE 5 By-reference only:

```
<element name="owner" type="gml:ReferenceType" minOccurs="0">
    <annotation>
        <appinfo>
            <gml:targetElement>ex:Person</gml:targetElement>
            <gml:reversePropertyName>ex:owns</gml:reversePropertyName>
        </appinfo>
    </annotation>
</element>
```

Depending on the encoding of the class, a UML attribute of a code list or enumeration type is mapped to an element with either a string value (value domain: values of the enumeration or code list) or a value referencing the corresponding dictionary entry. In an instance, the dictionary may be explicitly referenced using the `codeSpace` attribute. A default value for the URI representing the dictionary may be provided using an appinfo annotation element `gml:defaultCodeSpace`.

```
<element name="defaultCodeSpace" type="anyURI"/>
```

EXAMPLE 6 The code list "BuildingType" may be represented as:

```
<element name="type" type="ex:BuildingTypeType"/>
```

or

```
<element name="type" type="gml:CodeType">
    <annotation>
        <appinfo>
            <gml:defaultCodeSpace>http://www.someorg.de/example/cl.xml#BuildingType</gml:defaultCodeSpace>
        </appinfo>
    </annotation>
</element>
```

If a UML attribute or UML association role is redefined (i.e. a subclass contains an attribute or association role with the same name as in a supertype) then this property is not part of the content model of the subtype. It is the responsibility of an application to assert the compliance of instances with such constraints expressed in the conceptual model.

All attributes and association roles of a class shall be converted in the ascending sort order of the tagged value "sequenceNumber".

E.2.4.12 Documentation

Tagged values "documentation" from elements in the UML model are mapped to annotation/documentation elements in the XML Schema files.

EXAMPLE

```
<element name="curveProperty" type="gml:CurvePropertyType">
    <annotation>
        <documentation>This property element either references a curve via the XLink-attributes or contains the curve element. curveProperty is the predefined property which can be used by GML application schemas whenever a GML feature has a property with a value that is substitutable for AbstractCurve.</documentation>
    </annotation>
</element>
```

E.2.4.13 Classes imported from the ISO 19100 series of International Standards

In addition to the rules defined above, the following rules apply when the UML Application Schema imports classes from the ISO 19100 series of International Standards.

Classes from the ISO 19100 series of International Standards that are implemented by the GML schema shall be recognized. The use of classes from the ISO 19100 series of International Standards shall be conformant with ISO 19109. The mapping of the relevant classes from the ISO 19100 series of International Standards is shown in Table D.2.

If a class from ISO 19115 and implemented in ISO/TS 19139 is used as the type of a property, then an anonymous property type extending gml:AbstractMetadataPropertyType shall be defined. The encapsulated object element is the corresponding object element for the metadata type as specified by ISO/TS 19139.

E.2.4.14 Classes imported from other conceptual models with a predefined XML encoding

In addition to the rules defined above, the following rules apply when the UML Application Schema imports classes from another UML model for which a standard XML encoding has already been specified.

Extensions to Table D.2 for the imported classes shall be specified. The table shall be distributed together with the application schema in UML.

The mapping of the relevant classes from the imported model to XML Schema is normatively specified by this table.

E.3 Example <informative>

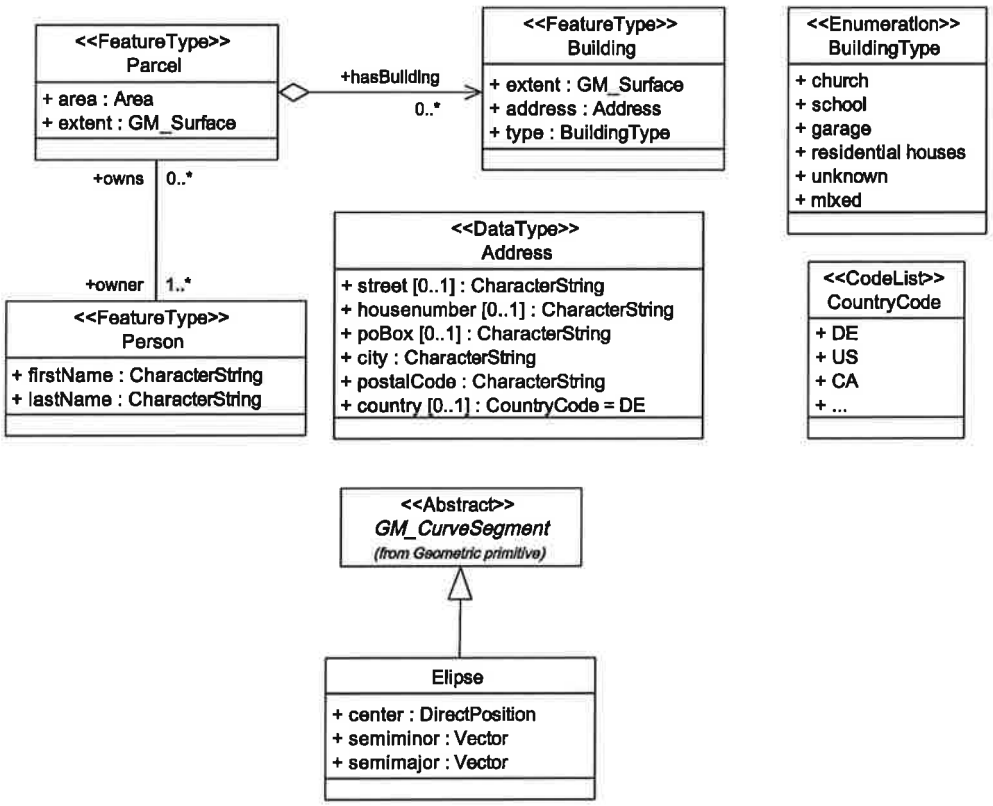

Figure E.7 — Example application schema

The application schema shown in Figure E.7 may be encoded as

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<schema targetNamespace="http://www.someorg.de/example" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ex="http://www.someorg.de/example" xmlns:gml="http://www.opengis.net/gml/3.2" elementFormDefault="qualified"
version="1.0">
    <!-- ================================== -->
    <import namespace="http://www.opengis.net/gml/3.2" schemaLocation="./gml.xsd"/>
    <import namespace="http://www.w3.org/1999/xlink" schemaLocation="./xlinks.xsd"/>
    <!-- ================================== -->
    <element name="Parcel" substitutionGroup="gml:AbstractFeature">
        <complexType>
            <complexContent>
                <extension base="gml:AbstractFeatureType">
                    <sequence>
                        <element name="area" type="gml:AreaType"/>
                        <element name="extent" type="gml:SurfacePropertyType"/>
                        <element name="owner" type="ex:PersonPropertyType" maxOccurs="unbounded">
                            <annotation>
                                <appinfo><gml:reverseProperty>ex:owns</gml:reverseProperty></appinfo>
                            </annotation>
                        </element>
                        <element name="hasBuilding" type="ex:BuildingPropertyType" minOccurs="0"
                            maxOccurs="unbounded"/>
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>
    <complexType name="ParcelPropertyType">
        <sequence minOccurs="0">
            <element ref="ex:Parcel"/>
        </sequence>
        <attributeGroup ref="gml:AssociationAttributeGroup"/>
        <attributeGroup ref="gml:OwnershipAttributeGroup" />
    </complexType>
    <!-- ================================== -->
    <element name="Building" substitutionGroup="gml:AbstractFeature">
        <complexType>
            <complexContent>
                <extension base="gml:AbstractFeatureType">
                    <sequence>
                        <element name="extent" type="gml:SurfacePropertyType"/>
                        <element name="address">
                            <complexType>
                                <sequence>
                                    <element name="Address" type="ex:AddressType"/>
                                </sequence>
                            </complexType>
                        </element>
                        <element name="type" type="ex:BuildingTypeType"/>
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>
    <complexType name="BuildingPropertyType">
        <sequence minOccurs="0">
            <element ref="ex:Building"/>
        </sequence>
        <attributeGroup ref="gml:AssociationAttributeGroup"/>
        <attributeGroup ref="gml:OwnershipAttributeGroup" />
    </complexType>
    <!-- ================================== -->
    <element name="Person" substitutionGroup="gml:AbstractFeature">
```

```
        <complexType>
            <complexContent>
                <extension base="gml:AbstractFeatureType">
                    <sequence>
                        <element name="firstName" type="string"/>
                        <element name="lastName" type="string"/>
                        <element name="owns" type="ex:ParcelPropertyType" minOccurs="0"
                        maxOccurs="unbounded">
                            <annotation>
                                <appinfo><gml:reverseProperty>ex:owner</gml:reverseProperty></appinfo>
                            </annotation>
                        </element>
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>
    <complexType name="PersonPropertyType">
        <sequence minOccurs="0">
            <element ref="ex:Person"/>
        </sequence>
        <attributeGroup ref="gml:AssociationAttributeGroup"/>
        <attributeGroup ref="gml:OwnershipAttributeGroup" />
    </complexType>
    <!-- ================================== -->
    <complexType name="AddressType">
        <sequence>
            <element name="street" type="string" minOccurs="0"/>
            <element name="housenumber" type="string" minOccurs="0"/>
            <element name="poBox" type="string" minOccurs="0"/>
            <element name="city" type="string"/>
            <element name="postalCode" type="string"/>
            <element name="country" type="ex:CountryCodeType" minOccurs="0" default="DE"/>
        </sequence>
    </complexType>
    <!-- ================================== -->
    <simpleType name="BuildingTypeType">
        <restriction base="string">
            <enumeration value="church"/>
            <enumeration value="school"/>
            <enumeration value="garage"/>
            <enumeration value="residential houses"/>
            <enumeration value="unknown"/>
            <enumeration value="mixed"/>
        </restriction>
    </simpleType>
    <!-- ================================== -->
    <simpleType name="CountryCodeType">
        <union memberTypes="ex:CountryCodeEnumerationType ex:CountryCodeOtherType"/>
    </simpleType>
    <simpleType name="CountryCodeEnumerationType">
        <restriction base="string">
            <enumeration value="DE"/>
            <enumeration value="US"/>
            <enumeration value="CA"/>
            <enumeration value="..."/>
        </restriction>
    </simpleType>
    <simpleType name="CountryCodeOtherType">
        <restriction base="string">
            <pattern value="other: \w{2,}"/>
        </restriction>
```

```
    </simpleType>
    <!-- ================================== -->
    <element name="Ellipse" type="ex:EllipseType" substitutionGroup="gml:AbstractCurveSegment"/>
    <complexType name="EllipseType">
        <complexContent>
            <extension base="gml:AbstractCurveSegmentType">
                <sequence>
                    <element name="center" type="gml:DirectPositionType"/>
                    <element name="semiminor" type="gml:VectorType"/>
                    <element name="semimajor" type="gml:VectorType"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</schema>
```

Annex F
(normative)

GML-to-UML application schema encoding rules

F.1 General concepts

The mapping from a GML application schema to an ISO 19109 conformant application schema in UML is based on a set of encoding rules. These encoding rules are conformant with the rules for GML application schemas as described in Clauses 7 to 21, especially Clauses 7, 9 and 21.

The rules listed in F.2 aim at an automatic mapping from a GML application schema to an ISO 19109 and ISO/TS 19103 conformant UML application schema.

These rules do not prescribe that all GML application schemas shall be generated to fulfil the encoding requirements documented in this annex. All schemas following the rules defined in Clause 21 are valid and conformant GML application schemas.

This annex shall be used if there is a requirement in the application domain to derive an ISO 19109 conformant Application Schema in UML from a GML application schema.

The XML namespace abbreviation "xsd" is used to refer to the namespace of XML Schema, which is "http://www.w3.org/2001/XMLSchema".

The XML namespace abbreviation "gml" refers to the XML namespace of GML, which is "http://www.opengis.net/gml/3.2".

In addition, GML imports definitions from the following namespaces:

The XML namespace abbreviation "xlink" refers to the XML namespace for xlinks, which is "http://www.w3.org/1999/xlink".

The term "GML namespaces" is used below to refer to the namespaces "gml" and "xlink".

F.2 Encoding rules

F.2.1 General encoding requirements

F.2.1.1 General remarks

The schema encoding rules are based on the general idea that the corresponding type and element declarations in XML Schema are mapped to class definitions in the UML application schema, so that element structures in the XML document can be mapped to the objects in the instance model.

F.2.1.2 GML schema

F.2.1.2.1 General

To be a valid input into the mapping, the GML application schema shall meet the requirements of the relevant conformance classes in 2.2, at least "All GML application schemas", "GML application schema to be converted to an ISO 19109 Application Schemas in UML" and "GML application schemas defining Features and Feature Collections".

The GML application schema shall have and contain definitions for only one target namespace.

The GML application schema may import definitions from XML namespaces other than its target namespace.

A GML application schema consists of a set of one or more XML schema documents such that:

— the documents have unique names;

— the documents contain xsd:include elements for other schema documents with the same target namespace;

— one top-level schema document for the GML application schema target namespace is not included by any other schema documents for the target namespace, but directly or indirectly includes all other schema documents for the target namespace, if any;

— the schema documents contain xsd:import elements for XML namespaces other than the target namespace, and for schema documents that contain definitions in those XML namespaces;

— all included and imported schema documents are accessible via the URI specified by the schemaLocation attribute on the xsd:include and xsd:import elements that reference them;

— a validating XML parser resolves all of the dependencies among the definitions contained in the set of schema documents;

— a validating XML parser validates the set of schema documents without error;

— a validating XML parser validates an XML instance document containing elements and attributes that represent all of the definitions from the target namespace of the GML application schema without error.

Documentation of the definitions contained in a GML application schema shall be stored in nested xsd:annotation and xsd:documentation elements within the schema definition elements.

The version of a GML application schema, if applicable, shall be contained in the version attribute of the xsd:schema element from the top-level schema for its target namespace.

All global type and element names within a GML application schema shall be unique.

The GML application schema shall not define any elements with anonymous types for objects.

The GML application schema shall not define any XML attributes or named groups.

Every complex type in a GML application schema shall either be a GML object type, a GML feature type, a GML data type or a GML property type.

Complex types with simple content shall not be defined in the GML application schema.

The name of all types defined in a GML application schema shall end with the suffix "Type".

A suffix "RestrictionType" in the name of a complex type shall only be used for an abstract type that derives by restriction and which is a the base type of exactly one complex type that derives from this type by extension and has the same name as the restricted type except that "RestrictionType" is replaced by "Type".

A suffix "PropertyType" in the name of a complex type shall only be used for an instantiable type that follows the pattern for by-reference-or-value property types of GML. A complex type (GML object type or GML feature type) with the same name shall exist that has "PropertyType" replaced by "Type".

A suffix "PropertyByValueType" in the name of a complex type shall only be used for an instantiable type that follows the pattern for by- value property types of GML. A complex type (GML data type, GML object type or GML feature type) with the same name shall exist that has "PropertyByValueType" is replaced by "Type".

NOTE These rules severely restrict the possible forms of GML application schemas.

F.2.1.2.2 GML object types including GML feature types

Each GML object type defined in a GML application schema shall have a content model that directly or indirectly derives from `gml:AbstractGMLType` and shall have a `gml:id` attribute.

Each GML object type of a particular kind defined in a GML application schema shall derive from the most specialized GML object type from the "http://www.opengis.net/gml/3.2" namespace of a similar kind (with matching semantics) that could possibly be used to define its content model. So GML object types defined in a GML application schema to represent geographic features (GML feature types) shall derive from `gml:AbstractFeatureType` instead of from `gml:AbstractGMLType`, GML object types defined in a GML application schema to represent geometric points shall derive from `gml:PointType` instead of from `gml:AbstractGeometryType`, etc.

GML object types defined in the GML application schema that derive from GML object types outside of the target namespace shall derive directly only from one of the GML object types listed in the third column of Table D.2 where there first column in the same row provides a class name of a class defined by the ISO 19100 series of International Standards or gml:AbstractGMLType or gml:AbstractFeatureType.

The schema definitions of abstract GML object types shall contain the attribute "abstract" with the value "true".

The name of abstract GML object types shall begin with the prefix "Abstract".

The schema definitions of GML object types for which no subtypes may be defined shall contain the attribute "final" with the value "all".

The properties of the GML object type shall be specified in an xsd:sequence element.

F.2.1.2.3 Global elements for gml object types

One global XML element shall be defined for every GML object type defined in a GML application schema.

The name of this element shall be the name of the GML object type without the "Type"-suffix.

The element shall have a substitutionGroup attribute whose value is the name of a global XML element whose type is the base type of the GML object type.

F.2.1.2.4 Default property types for gml object types

A default GML property type may be defined in a GML application schema for every GML object type defined in that GML application schema.

The GML property type shall either use or inherit directly or indirectly from one of the property types specified in 7.2.3 or it shall be defined in accordance with the patterns specified in this subclause.

The name of this property type shall be the name of the GML object type with the "Type"-suffix replaced by "PropertyType".

If no default property type is specified for a GML object type, an application schema shall use `gml:ReferenceType` as the default property type of the GML object type.

F.2.1.2.5 Inline property types for gml object types

A default GML property type for inline properties may be defined in a GML application schema for every GML object type defined in that GML application schema.

The GML property type shall either inherit directly or indirectly from `gml:InlinePropertyType`, or it shall be defined in accordance with the patterns specified in 7.2.3.8. The use of the `gml:AsscociationAttributeGroup` is prohibited in such properties.

The name of this property type shall be the name of the GML object type with the "Type"-suffix replaced by "PropertyByValueType".

If no default property type for inline properties is specified for a GML object type, an application schema shall use `gml:AssociationRoleType` as the default property type for inline properties of the GML object type.

F.2.1.2.6 GML data types including GML union types

A complex type defined in a GML application schema that does not directly or indirectly derive from `gml:AbstractGMLType` is called a GML data type.

The properties of the GML data type shall take one of the following forms:

— The properties of the complex type as well as the properties of all of its base types are specified in an xsd:sequence element with minOccurs and maxOccurs values of "1".

— The GML data type is not derived from any base type. In this case, the properties may be specified in either a single xsd:sequence element with minOccurs and maxOccurs values of "1" or a single xsd:choice element with minOccurs and maxOccurs values of "1".

The content model of the complex type shall not include a `gml:id` attribute.

F.2.1.2.7 Default property types for GML data types

A default GML property type for inline properties may be defined in a GML application schema for every GML data type defined in that GML application schema.

The GML property type shall either inherit directly or indirectly from `gml:InlinePropertyType`, or it shall be defined in accordance with the patterns specified in 7.2.3.8. The use of the `gml:AsscociationAttributeGroup` is prohibited in such properties.

The name of this property type shall be the name of the GML data type with the "Type"-suffix replaced by "PropertyByValueType".

If no default property type for inline properties is specified for a GML data type, an application schema shall use `gml:AssociationRoleType` as the default property type for inline properties of the GML data type.

F.2.1.2.8 Enumerations

A simple type defined in a GML application schema that is a restriction of xsd :string using only the xsd:enumeration facet is called an enumeration.

F.2.1.2.9 Code lists

A simple type defined in a GML application schema that is a union of an enumeration and a simple type that is a restriction of xsd:string using only one xsd:pattern facet with the value "other: \w{2,}" is called a code list.

Enumeration values may be qualified with an appInfo annotation (element `gml:codeListValue`) specifying that the enumeration value is the code value of another enumeration value; the associated enumeration value is given as the text value of the `gml:codeListValue` element.

F.2.1.2.10 Global elements for GML data types, enumerations and code lists

No global XML element shall be defined for enumerations or code lists defined in a GML application schema.

F.2.1.2.11 Predefined basic types

The simple types from the XML Schema and GML namespace listed in the fourth column of Table D.2 may be used in the GML application schema. No other simple types from these namespaces shall be used in a GML application schema.

F.2.1.2.12 GML properties

Every property of a GML object or feature type (except properties defined in the GML namespace) or of a GML data or union type shall be represented by a single, locally defined xsd:element. Locally defined means that the name and type of the element shall be given explicitly in the element declaration (no references to global XML elements). The element may carry minOccurs and maxOccurs values. The name of this element shall be the name of the property; the type shall be either a simple type or a property type.

F.2.1.2.13 Schematron constraints

All Schematron constraints are ignored.

F.2.1.2.14 Imported elements and types from other XML namespaces

If other XML Schema components are imported from other namespaces than XML Schema and GML, define the relevant entries as extensions to Table D.2.

F.2.1.2.15 Other information

All other information in the GML application schema is not used in the encoding rules and is ignored.

F.2.1.3 Character repertoire and languages

The character encoding used for the schemas determines the available character repertoire.

F.2.1.4 Exchange metadata

Exchange metadata may be specified for every Feature or Feature Collection in a GML instance document[11]. No specific schema for the exchange metadata is added to the GML application schema.

F.2.1.5 Dataset and object identification

Unique `gml:id` identifiers in accordance with 7.2.4.5 and XML's ID mechanism shall be used to identify GML objects.

F.2.1.6 Update mechanism

No explicit update mechanism shall be defined for the feature types defined in a GML application schema. It is assumed that other mechanisms are used to update an instance model data store.

F.2.1.7 Input data structure

The schema for the input data structure is defined by the XML Schema 1.0 Part 1: Structures, Part 2: Datatypes W3C Recommendations, and the Rules for GML application schemas (see Clause 21)

F.2.2 Output data structure

See ISO 19118:2005, A.3, for a description of the output data structure.

F.2.3 Conversion rules

F.2.3.1 General concepts

The schema conversion rules defined in the following subclauses describe the mapping from a GML application schema that follows the guidelines described in F.2.1 to a UML application schema that conforms to the rules defined in ISO 19109 and ISO/TS 19103, using the encoding rules of ISO 19118:2005, Annex A, and in particular the generic instance model described in A.3. These rules are also based on the current rules for the GML model and syntax as described in Clauses 7 to 21 (especially Clause 7).

The schema conversion rules map definitions from a (set of) valid GML application schema documents (XSDs) to a set of UML packages. A top-level package with the stereotype <<Application Schema>> is created to contain all the other packages in this set. By default, one package is created in this set for each XSD in the GML application schema, including those directly or indirectly imported from XML namespaces other than the target namespace for the GML application schema, except for XSDs for the GML namespaces. The top-level package owns directly or indirectly all UML model elements mapped from object types in the GML application schema.

The declarations of the GML application schema may be arranged in a different package structure as long as the top-level package keeps its name and stereotype and all the model elements still belong directly or indirectly to this package.

---

11) By using the property elements whose content model has been derived from `gml:AbstractMetadataPropertyType` and, for example, the ISO/TS 19139 XML Schema encoding of ISO 19115:2003.

The type and element declarations in the GML application schema are mapped to class definitions in the UML application schema, so that element structures in the GML XML document can be mapped to corresponding objects in the instance model.

The UML model shall contain within a package with the name "ISO 19100" the applicable normative packages of the ISO 19100 series of International Standards or a strict profile of this model.

The UML model shall contain the UML package of all other GML application schemas imported by the GML application schema.

Table F.1 gives an overview; full details of the mapping are specified in the subsequent subclauses.

Table F.1 — Schema encoding overview

Table: GML → UML Application Schema Overview	
GML application schema	**UML application schema**
GML application schema	Package <<ApplicationSchema>>
GML schema document {name} XSD	Package named {name}
Object and property type and global element for any object type that is a direct or indirect extension of `gml:AbstractFeatureType`	Class with stereotype <<FeatureType>>
Object and property type and global element for any object type that is a direct or indirect extension of `gml:AbstractGMLType`, other than those that extend `gml:AbstractFeatureType`	Class with no stereotype
Data and property type and global element for any object type that is not a direct or indirect extension of `gml:AbstractGMLType` and whose content model is a sequence of properties	Class with stereotype <<DataType>>
Restriction of `xsd:string` with enumeration values	Class with stereotype <<Enumeration>>
Union of an enumeration and a pattern	Class with stereotype <<CodeList>>
Data and property type and global element for any object type that is not a direct or indirect extension of `gml:AbstractGMLType` and whose content model is a choice of properties	Class with stereotype <<Union>>
Local xsd:element of a simpleType or a complexType with simpleContent or a type that does not directly or indirectly inherit from `gml:AbstractGMLType`	UML Attribute
Local xsd:element of a type that contains `gml:AssociationAttributeGroup`	UML Association Role
Schematron constraints	Not encoded

The multiplicity of attributes and association roles is derived from the minOccurs and maxOccurs attributes in local xsd:element declarations.

F.2.3.2 GML schema documents

A top-level package with the stereotype <<Application Schema>> is created to contain all the other packages generated for the GML application schema.

— The "targetNamespace" and "xmlns" tagged values are applied to the <<ApplicationSchema>> package with corresponding values for the target namespace of the GML application schema

EXAMPLE "http://www.myorg.com/myns" and "myns".

— The "version" tagged value is applied to the <<ApplicationSchema>> package with the default value of "1.0". If the "version" attribute of the xsd:schema element of the top-level schema document for the GML application schema exists and contains a non-empty value, its value replaces the default tagged value.

— The "xsdDocument" tagged value is set to the relative filename of the XML Schema document.

By default, one UML package is generated for each input schema document in the GML application schema, including those directly or indirectly imported from XML namespaces other than the target namespace of the GML application schema — except for XML Schema documents from the GML namespaces. Alternatively, a single XML Schema document may also be split into several UML packages.

The packages are generated in the <<ApplicationSchema>> package for the GML application schema with names that correspond to the names of the input schema documents.

The xsd:include and xsd:import statements in each input schema document are used to determine and set the dependencies of the packages generated in the <<Application Schema>> package.

F.2.3.3 GML object types

Every GML object type shall be mapped to a UML class.

If the object type directly or indirectly derives from `gml:AbstractFeatureType`, the stereotype of the class shall be <<FeatureType>>, otherwise no stereotype shall be set.

The name of the class shall be the same as the name of the global element of the GML object type.

The class shall be abstract, if and only if the GML object type is abstract.

If the GML object type is derived from another GML object type, then the class inherits from the corresponding superclass. If the base type is defined in the GML application schema or another imported GML application schema, then the superclass is the class corresponding to this GML object type. If the base type is defined in the GML namespace, then the superclass is determined by Table D.2. If the base type is listed in the third column of that table, then the superclass is the class in the first column of the same row.

The GML properties of the GML object type shall be mapped to attributes and association roles as described in F.2.3.9. Assign a tagged value "sequenceNumber" to all UML attributes and association roles created in this mapping with unique integer values in ascending order reflecting the order of the properties in the sequence of the object type.

F.2.3.4 GML object types (imported from the GML schema)

The complex types from the GML namespace listed in the left hand column of Table D.2 shall be mapped to the predefined UML classes implemented by the ISO geographic information standards profile of GML in the second column of the table.

F.2.3.5 Basic types

The simple types from the XML Schema and GML namespace shown in the right hand column of Table D.2 shall be mapped to the predefined UML classes implemented by the ISO geographic information standards profile of GML in the left hand column of the table.

F.2.3.6 GML data types

Every GML data type shall be mapped to a UML class. The name of the class shall be the same as the name of the complex type without the "Type"-suffix.

If the GML data type is derived from another GML data type (base type), then the class inherits from the corresponding superclass.

If the properties of the GML data type are embedded in an xsd:sequence element, the stereotype of the class shall be <<DataType>>, if they are embedded in an xsd:choice element, the stereotype of the class shall be set to <<Union>>.

The GML properties of the GML object type shall be mapped to attributes and association roles as described in F.2.3.9. Assign a tagged value "sequenceNumber" to all UML attributes and association roles created in this mapping with unique integer values in ascending order reflecting the order of the properties in the sequence of the object type.

F.2.3.7 Enumerations

A simple type defined in the GML application schema as a restriction of xsd:string with enumeration values shall be mapped to a class with the <<Enumeration>> stereotype in the UML application schema.

The name of the class shall be the name of the simple type.

Every xsd:enumeration facet without an xsd:appInfo annotation with a child element `gml:codeListValue` shall be mapped to a UML attribute with the value as the attribute name.

Every xsd:enumeration facet with an xsd:appInfo annotation with a child element `gml:codelistValue` shall be mapped to an initial value of the UML attribute with the same name as the value of the `gml:codelistValue` element. If no such UML attribute exists in the class, the facet shall be ignored.

F.2.3.8 Code lists

A simple type defined an the GML application schema as a union of an xsd:pattern restriction with the value "other:\w{2,}" and an enumeration shall be mapped to a class with the stereotype <<CodeList>> in the UML application schema.

The name of the class shall be the name of the simple type.

Every xsd:enumeration facet of the enumeration without an xsd:appInfo annotation with a child element `gml:codelistValue` shall be mapped to a UML attribute with the value as the attribute name.

Every xsd:enumeration facet of the enumeration with an xsd:appInfo annotation with a child element `gml:codelistValue` shall be mapped to an initial value of the UML attribute with the same name as the value of the `gml:codelistValue` element. If no such UML attribute exists in the class, the facet shall be ignored.

F.2.3.9 GML properties

If the type of a property element:

— is a simple type or the property type of GML data type, the property shall be mapped to a UML attribute with the corresponding type as the data type;

— is a property type of a GML object type (inline and/or by-reference) whose content model is directly or indirectly derived from gml:AbstractMemberType, the property shall be mapped to a UML association role of a UML aggregation to the class representing the target GML object type; if the content model of the property element contains an attribute "owns" with a fixed value of "true" (through a Schematron constraint) then the UML aggregation shall be change to a UML composition;

— is a property type of a GML object type (inline and/or by-reference), the property shall be mapped to a UML association role of a UML association to the class representing the target GML object type; if the property type supports only by-reference, the target GML object type shall be determined from the embedded xsd:appInfo annotation with a child element `gml:targetElement` specifying the qualified element name of the target type. The tagged value "inlineOrByReference" shall be set to "inline" for representations that allow only an inline encoding of the property value and to "byReference" for representations that allow only a by-reference encoding of the property value;

— is a property type of a GML object type (inline and/or by-reference) whose content model is directly or indirectly derived from gml:AbstractMetadataPropertyType, the UML attribute or association role shall carry a tagged value "isMetadata" with the value "true".

The name of the UML attribute or association role shall be the name of the GML property element.

The multiplicity of the UML attribute or association role shall be derived from the minOccurs and maxOccurs value of the GML property.

If the property element has an xsd:appInfo annotation with a child element `gml:reversePropertyName` embedded, then the association role shall be defined as part of the association between the two classes where the other association role has a name equal to the value of the `gml:reversePropertyName` element.

F.2.3.10 Documentation

XML Schema xsd:annotation/xsd:documentation elements in GML application schemas are mapped to "documentation" tagged values in the UML application schema.