TECHNICAL SPECIFICATION

ISO/TS 19139-1

First edition
2019-03

Geographic information — XML schema implementation —

Part 1: Encoding rules

Reference number
ISO/TS 19139-1:2019(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

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

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 211, *Geographic information/Geomatics*.

This first edition of ISO/TS 19139-1 cancels and replaces ISO/TS 19139:2007, which has been technically revised.

A list of all parts in the ISO 19139 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

The importance of metadata describing digital geographic data is explained in detail in the text of ISO 19115-1, and other International Standards, e.g. ISO 19110, ISO 19119, ISO 19157. Those documents provide a structure for describing digital geographic data by defining metadata elements and establishing a common set of metadata terminology, definitions and extension procedures. These standards do not define encodings for those metadata.

To facilitate the standardization of implementations across the standards and in similar domain schemas, this document provides a definitive set of rules for encoding ISO metadata standards in Extensible Markup Language (XML). The resulting XML schemas are meant to enhance interoperability by providing a common specification for describing, validating and exchanging metadata. These rules are intended to be used in parallel to the rules in ISO 19136:2007, Annex E for encoding application schemas into XML/GML. The difference is that those rules are for data that represents features; these rules are for metadata about that data.

ISO 19118 describes the requirements for creating encoding rules based on UML schemas and the XML based encoding rules as well as introducing XML. This document uses the encoding rules defined in ISO 19118 and provides the specific details of their application with regards to deriving XML schema for the UML models for other metadata standards.

These rules were first used in creating ISO/TS 19115-3 as an XML encoding of ISO 19115-1, i.e. ISO/TS 19115-3 conforms to this document. They were also used to create ISO/TS 19157-2, an encoding of ISO 19157.

The standardization target of this document is XML implementations of metadata. This includes both other standards within the Geographic Information series and models developed by other organizations.

Geographic information — XML schema implementation —

Part 1:
Encoding rules

1 Scope

This document defines XML based encoding rules for conceptual schemas specifying types that describe geographic resources. The encoding rules support the UML profile as used in the UML models commonly used in the standards developed by ISO/TC 211. The encoding rules use XML schema for the output data structure schema.

The encoding rules described in this document are not applicable for encoding UML application schema for geographic features (see ISO 19136 for those rules).

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 19118, *Geographic information — Encoding*

W3C XMLName, Namespaces in XML. W3C Recommendation

W3C XMLSchema-1, XML Schema Part 1: Structures. W3C Recommendation

W3C XMLSchema-2, XML Schema Part 2: Datatypes. W3C Recommendation

W3C XML, Extensible Markup Language (XML) 1.0, W3C Recommendation

W3C XLink, XML Linking Language (XLink) Version 1.0. W3C Recommendation

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
namespace
collection of names, identified by a URI reference, which are used in XML documents as element names and attribute names

[SOURCE: W3C XML]

3.2
package
<UML> general purpose mechanism for organizing elements into groups

EXAMPLE Identification information, metadata entity set information, constraint information.

[SOURCE: ISO 19103:2015, 4.27 — modified: The EXAMPLE has been added.]

3.3
polymorphism
characteristic of being able to assign a different meaning or usage to something in different contexts, specifically, to allow an entity such as a variable, a function, or an object to have more than one form

Note 1 to entry: There are several different kinds of polymorphism.

[SOURCE: https://searchcio.techtarget.com/]

3.4
realization
<UML> specialized abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other representing an implementation of the latter (the client)

Note 1 to entry: Realization indicates inheritance of behaviour without inheritance of structure.

[SOURCE: ISO 19103:2015, 4.29]

4 Symbols and abbreviated terms

4.1 Abbreviated terms

UML	Unified Modeling Language
URI	Uniform Resource Identifier
XCT	XML Class Type
XCPT	XML Class Property Type
XCGE	XML Class Global Element
XML	Extensible Markup Language
XSD	XML Schema Definition

4.2 Namespace abbreviations

Table 1 presents the external namespaces used in this document. The left column shows the common namespace prefix used to describe the elements in the namespace. The second column shows the English description of the namespace prefix. The third column is the URI of the actual namespace. These URIs do not correspond necessarily to the location of the schemas.

Table 1 — External namespaces used by this document

Namespace prefix	English description of the namespace	URI of the actual namespace
gml	Geography Markup Language	http://www.opengis.net/gml/3.2
xlink	XML Linking Language (XLink)	http://www.w3.org/1999/xlink
xs	W3C XML base schemas	http://www.w3.org/2001/XMLSchema

4.3 UML model stereotypes

4.3.1 Overview of UML model stereotypes

A UML stereotype is an extension mechanism for existing UML concepts. In addition to the stereotypes already defined for describing geographic resources, this document defines stereotypes necessary for a rules-based encoding into XML schema.

The elements of the UML diagrams depicted in Clause 7 can carry stereotypes specifying an XML implementation. Those stereotypes, listed in the following subclauses, are carried by classes representing XML elements or XML types, UML attributes, UML links (realizations or dependencies) and UML packages.

4.3.2 Stereotypes of classes

In this document the following Stereotypes of classes are used:

a) <<xs:choice>>: The class represents an implementation type encoded as an XML choice block. Each property of the class is implemented as an element of the choice.

b) <<xs:complexType>>: The class represents an implementation type encoded as an XML complex type.

c) <<xs:simpleType>>: The class represents an implementation type encoded as an XML simple type.

d) <<xs:simpleContent>>: The class represents an implementation type encoded as an XML complex type with simple content.

4.3.3 Stereotypes of attributes

In this document the following Stereotypes of attributes are used:

a) <<xs:attribute>>: The property is encoded as an XML attribute.

b) <<xs:attributeGroup>>: The property is encoded as an XML attributeGroup.

c) <<xs:element>>: The property is encoded as an XML element with a name and a type (<xs:element name=”propertyName” type=”propertyType”/>).

4.3.4 Stereotypes of links

In this document, the following Stereotypes of links are used:

a) <<XCT>>: (carried by realization relationships) The XCT of the abstract concept to implement is substituted by the specified external implementation.

b) <<XCGE>>: (carried by realization relationships) The XCGE of the abstract concept to implement is substituted by the specified external implementation.

c) <<XCPT>>: (carried by realization relationships) The XCPT of the abstract concept to implement is substituted by the specified external implementation.

d) <<implement>>: (carried by dependency relationships) The source represents an XML schema implementing the abstract concepts defined in the target.

e) <<include>>: (carried by dependency relationships) The source and the target represent XML schemas. The source includes (<xs:include … />) the target.

f) <<import>>: (carried by dependency relationships) The source and the target represent sets of XML objects grouped within the same namespace. The source imports (<xs:import … />) the target.

4.3.5 Stereotypes of packages

In this document, the following Stereotypes of packages are used:

a) <<xmlSchema>>: The package represents an XML schema.

b) <<xmlNamespace>>: The package represents a set of XML objects grouped within the same namespace.

5 Conformance

The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance, are specified in ISO 19105. An XML schema implementation of geographic resources conforms to this document if it passes the test modules defined in Annex A.

6 Requirements for encoding

6.1 Overview of requirements

Geographic resources are generally specified in other standards and specifications, e.g. metadata according to ISO 19115-1 and feature catalogues according to ISO 19110. Geographic resources are represented as conceptual schemas comprising a set of UML packages containing one or more UML classes. These conceptual schemas provide an encoding-independent view of the related geographic resources.

XML offers many alternatives for structuring information for exchange. Clause 7, in conformance with ISO 19118, defines XML schema encoding rules more specifically applicable to the conceptual schemas of geographic resources.

Even within the reduced limitations of ISO 19118 and the encoding rules defined in this document, there are still choices for the creation of specific XML schemas.

This document provides the foundation needed to establish consistent XML schema implementations of geographic resources, in the form of a set of standardized encoding rules to be applied to the conceptual schemas of geographic resources to create standard XML schema.

The details of XML namespaces are not included in this document. A *namespace* is a collection of names that can be used in XML documents as element or attribute names. The namespace identifier is used to identify the names with a specific schema. A namespace identifier is a URI. A URI is often cumbersome for reading, writing and including in human discussion, so this document will refer to common namespace prefixes when identifying the contents of a namespace.

Before presenting the details of the encoding, it is important to understand why certain encoding rules are used. Gaining an understanding of the rules will make clear the capabilities, limitations and best-practice use. Some of the major goals considered when developing this encoding rule were interoperability with other ISO 19100 series specifications, predictability, extensibility and usability. Additional details of these goals are described in 6.2 to 6.6.

6.2 Rule-based

This document defines a rule-based encoding built from UML models, e.g. those in the ISO 19100 series of International Standards, as required by ISO 19118. Using a rule-based method achieves three important goals:

— first, the resulting XML schemas are based directly on the conceptual models and therefore increase the chance for interoperability;

— second, the resulting schema is predictable since any class, attribute, association, etc. is encoded just as all other UML elements of the same type are encoded;

— third, XML schemas using these rules can be generated in an automated or semi-automated fashion.

6.3 Quality

XML schema quality in the context of this document implies that all elements of the conceptual model are consistently implemented and that a user can directly create and/or understand the content of an XML instance document using the conceptual schema of the corresponding geographic resource. Additionally, an implementer can determine the XML schema implementation of a geographic resource by knowing its conceptual schema and the encoding rules.

Another aspect to quality is completeness. This document enables encoding the entire conceptual schema of a geographic resource without regard to usage or application.

6.4 Web implementations

One of the goals stated in 6.1 is usability. Usability, as it pertains to the design of geographic resources, focuses on their exchange with the understanding that this will often happen in a web-based environment. While there is no restraint against creating instance documents that never transfer across a network, there are many aspects to the design that are intended to aid Internet and web-like transfer of geographic resources.

6.5 Use of external XML implementations

Another design principle that aids interoperability and usability is the re-use of existing XML schemas. If an XML schema standard already exists that encodes a part of the ISO 19100 series pertaining to geographic metadata, then it is advantageous to incorporate that XML schema standard. If the external XML schema is directly used, then interoperability is enhanced. It is also likely that software already exists that can process instance documents that conform to the external XML. Furthermore, if the external schema is well designed, it might be more efficient than XML schema generated from a series of encoding rules and this might help achieve the goal of usability.

While using an implementation that already exists has some important advantages, it is recommended that the external XML schemas should not violate the primary design principles defined herein. See 7.3.8 for details and examples.

6.6 Polymorphism

The term polymorphism is formally defined in 3.3. In general terms, polymorphism means the ability to assume different forms, i.e. to implement properties using their data type or any of its derived classes. Polymorphism allows implementers to extend the more general format of *properties* within their namespaces while still providing usable and understandable instance documents for users outside of their organization. Polymorphism primarily derives from the property type encodings described in 7.3.

7 Encoding rules

7.1 Overview of encoding rules

General rules for transforming UML to XML schema are described below and are in accordance with the rules defined in ISO 19118. In some cases, ISO 19118 allows for multiple methods of transforming UML to XML schema, and the rules defined here serve to clarify which method is to be used to conform with this document. Background on classes and how they are the building blocks for encoding all data exchange (and in this case metadata exchange) is purposefully absent from this document since ISO 19118 covers this in detail. This document is based on the encoding rules in ISO 19118 and a familiarity with that document will greatly enhance comprehension of the topics described throughout Clause 7.

NOTE The way the encoding rules are described doesn't prevent adopting best practices in generating XML schemas.

7.2 Default encoding

7.2.1 XML class type (XCT)

7.2.1.1 Overview of XML class type (XCT)

The class is the fundamental modelling concept in UML (ISO 19118), so the fundamental encoding rules focus on the encoding of a UML class and build from there. A class is made up of one or more properties. It is important to recall from ISO 19118 that a property can represent an attribute, association, aggregation or composition (ISO 19118). For example, in Figure 1, the *Class1* class has three properties: *attr1*, *attr2* and *role1*. For the sake of encoding into XML schema it is important to understand that there is no distinction between properties that are UML attributes, associations, aggregations or compositions.

ISO 19118 also describes the need to use identifiers (ids) and domain unique identifiers (DUIDs) as identifiers in XML schema. There is a special XML schema type in baseTypes2014.xsd in the gco namespace, gco:AbstractObject_Type, which provides the necessary identifiers. It is mentioned here because it is part of the default XML Class type encoding.

7.2.1.2 XCT rule

As a worked example, the UML in Figure 1 is encoded in XML using the following requirements.

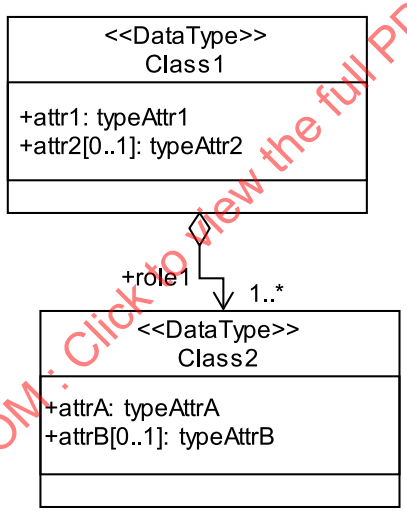

Figure 1 — Example

Requirement /req/default/XCT

A UML Class shall be encoded into XML schema as an XML complex type (xs:complexType) referred to as the XML Class Type (XCT).

Requirement /req/default/XCT-name

Each XCT shall have a name attribute whose value is the class name with the suffix _Type:

EXAMPLE 1 Step 1) of building the Class1 class shown in Figure 1 is:

```
<xs:complexType name="Class1_Type">
        (...)
</xs:complexType>
```

Requirement /req/default/XCT-complex-content

All UML Classes following the default encoding rules shall have complex content, and to provide this capability, the xs:complexType element contains an xs:complexContent element.

EXAMPLE 2 Step 2) of building the Class1 class shown in Figure 1 is:

```
<xs:complexType name="Class1_Type">
  <xs:complexContent>
       (...)
  </xs:complexContent>
</xs:complexType>
```

Requirement /req/default/XCT-extend-abstract

All UML Classes following the default encoding rules shall extend the gco:AbstractObject_Type which is done by adding an xs:extension element with the base attribute equal to gco:AbstractObject_Type.

EXAMPLE 3 Step 3) of building the Class1 class shown in Figure 1 is:

```
<xs:complexType name="Class1_Type">
  <xs:complexContent>
    <xs:extension base="gco:AbstractObject_Type">
       (...)
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Requirement /req/default/XCT-sequence

All UML Classes following the default encoding rules shall have a sequence containing all the properties of the class. This is accomplished by adding an xs:sequence element, containing xs:element elements for each property of the class.

Requirement /req/default/XCT-properties

The attributes of the *xs:element* element shall be:

a) the *name* attribute, equal to the name of the property;

b) the *type* attribute, equal to the name of the XCPT corresponding to the UML class specified as the type of the property. Subclause 7.2.3 explains that by default this is the class name plus "_PropertyType" and 7.3.8 defines the only exceptions to this XCPT naming convention. The name of the XCPT used as the value for this type attribute will also be properly prefixed with the appropriate namespace.

c) the *minOccurs* and *maxOccurs* attributes, with the values described in ISO 19118:2011, Table C.5. Additionally, if a property of the class happens to be an attribute that uses the 'set' or 'sequence' structure, then the *minOccurs* attribute shall be "0" for optional attributes and "1" for mandatory attributes and the *maxOccurs* is "unbounded".

EXAMPLE 4 Step 4) of building the Class1 class shown in Figure 1 is:

```
<xs:complexType name="Class1_Type">
  <xs:complexContent>
    <xs:extension base="gco:AbstractObject_Type">
      <xs:sequence>
        <xs:element name="attr1" type="ns1:typeAttr1_PropertyType"/>
        <xs:element name="attr2" type="ns1:typeAttr2_PropertyType" minOccurs="0" />
         <xs:element name="role1" type="ns1:Class2_PropertyType" minOccurs="1"
            maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

NOTE Throughout the examples in this document there is a namespace prefix, “ns1” that is not always shown in the UML. This fictitious namespace is used to illustrate when a namespace prefix should be present but it does not correspond to any of the namespaces or corresponding prefixes defined in this document.

Requirement /req/default/XCT-properties-sequence

The order of the xs:elements within the xs:sequence shall be the same as the order of the property’s corresponding entry in the data dictionary documenting the conceptual schema.

This requirement applies to default encodings as well as all special case encodings for XCTs.

7.2.1.3 XCT rule example

Applying the XCT Rule above to this UML example (from ISO 19157) results in the XML below.

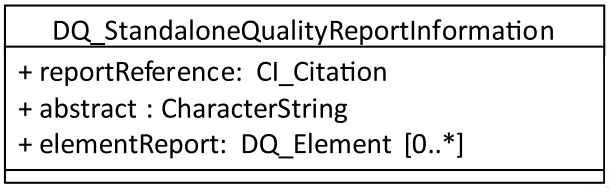

Figure 2 — StandAloneQualityReport information UML from ISO 19157:2013, 10.1

The XCT corresponding to Figure 2:

```
<xs:complexType name="DQ_StandaloneQualityReportInformation_Type">
   <xs:complexContent>
     <xs:extension base="gco:AbstractObject_Type">
        <xs:sequence>
          <xs:element name="reportReference" type="mcc:Abstract_Citation_PropertyType"/>
          <xs:element name="abstract" type="gco:CharacterString_PropertyType"/>
          <xs:element name="elementReport" type="mdq:AbstractDQ_Element_PropertyType"
          maxOccurs="unbounded" minOccurs="0" />
        </xs:sequence>

        </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

7.2.2 XML Class Global Element (XCGE)

7.2.2.1 Overview of the XML Class Global Element (XCGE)

The global element is defined so that it can be referenced from other sections of the schema.

7.2.2.2 XCGE rule

Requirement /req/default/XCGE

A UML Class shall also be encoded as a global element with a name attribute equal to the name of the UML class and the type equal to the name of the XCT described in 7.2.1 (with the appropriate namespace prefix included). Hereafter, this global element is referred to as the XML Class Global Element (XCGE).

EXAMPLE 1 The XCGE of the Class1 class shown in Figure 1 is:

```
<!-- ..................................................... -->
<xs:element name="Class1" type="ns1:Class1_Type"/>
<!-- ..................................................... -->
```

7.2.2.3 XCGE example

EXAMPLE 2 The XCGE corresponding to DQ_DataQuality shown in Figure 2 is:

```
<!-- ..................................................... -->
<xs:element name="DQ_StandaloneQualityReportInformation"
   substitutionGroup="gco:AbstractObject"
   type="mdq:DQ_StandaloneQualityReportInformation_Type"/>
<!-- ..................................................... -->
```

7.2.3 The XML Class Property Type (XCPT)

7.2.3.1 Overview of the XML Class Property Type (XCPT)

The XCPT rule is to support the property concept described in 7.2.1 and the potential for a given class to be a property type for a set of container classes

7.2.3.2 XCPT rule

Requirement /req/default/XCPT

To support the property concept described in 7.2.1 and the potential for a given class to be a property type for a set of container classes, each UML class shall also be defined as another XSD xs:complexType, which is hereafter referred to as the XML Class Property Type (XCPT). Recognize that the containment of a property is managed through the XCPT of its data type. By default, both “By value” and “By Ref” containment are allowed by the XCPT. The following list details the steps for encoding an XCPT, for a default class.

Requirement /req/default/XCPT-name

a) To distinguish the XCPTs from the XCTs, the xs:complexType element shall have a name attribute equal to the class name suffixed with _PropertyType.

EXAMPLE 1 Step 1) of building the Class1 class XCPT shown in Figure 1 is:

```
<xs:complexType name="Class1_PropertyType">
    (...)
</xs:complextype>
```

Requirement /req/default/XCPT-sequence

b) To provide "by Value" containment in a manner that supports the polymorphism requirements described in 6.6, the XCPT shall have an optional xs:sequence element that contains one xs:element element with a ref attribute equal to the XCT of the UML. The xs:sequence element is made optional, in order to allow for the possibility that the property will be implemented "by Ref".

EXAMPLE 2 Step 2) of building the Class1 class XCPT shown in Figure 1 is:

```
<xs:complexType name="Class1_PropertyType">
  <xs:sequence minOccurs="0">
    <xs:element ref="ns1:Class1"/>
  </xs:sequence>
</xs:complexType>
```

Requirement /req/default/XCPT-reference

c) To provide "by Ref" containment, the XCPT shall have an xs:attributeGroup element with a ref attribute equal to "gco:ObjectReference".

EXAMPLE 3 Step 3) of building the Class1 class XCPT shown in Figure 1 is:

```
<xs:complexType name="Class1_PropertyType">
  <xs:sequence minOccurs="0">
    <xs:element ref="ns1:Class1"/>
  </xs:sequence>
  <xs:attributeGroup ref="gco:ObjectReference"/>
</xs:complexType>
```

Requirement /req/default/XCPT-nilReason

d) To provide Null value explanations when necessary, the XCPT shall have an xs:attribute element with a ref attribute equal to “gco:nilReason”. The gco:nilReason XML attribute manages null values in an XML instance document. At the property level, this attribute allows a reason (explaining why the actual value cannot be provided) to exist in place of an actual value. The corresponding XML schema fragment is shown below:

XML attribute	<xs:attribute name="nilReason" type="gml:NilReasonType"/>

The gml:NilReasonType is fully described in ISO 19136 and is an enumerated XML type allowing the values: “inapplicable”, “missing”, “template”, “unknown”, and “withheld”.

EXAMPLE 4 Step 4) of building the Class1 class XCPT shown in Figure 1 is:

```
<xs:complexType name="Class1_PropertyType">
  <xs:sequence minOccurs="0">
    <xs:element ref="ns1:Class1"/>
  </xs:sequence>
  <xs:attributeGroup ref="gco:ObjectReference"/>
  <xs:attribute ref="gco:nilReason">
</xs:complexType>
```

7.2.3.3 XCPT example

EXAMPLE 5 Sample XCPT corresponding to DQ_StandAloneReportInformation shown in Figure 2.

```
<xs:complexType name="DQ_StandaloneQualityReportInformation_PropertyType">
   <xs:sequence minOccurs="0">
      <xs:element ref="mdq:DQ_StandaloneQualityReportInformation"/>
   </xs:sequence>
   <xs:attributeGroup ref="gco:ObjectReference"/>
   <xs:attribute ref="gco:nilReason"/>
</xs:complexType>
```

Notice that there is no specific constraint introduced in the XCPT to force the containment information of a property defined in UML. The UML containment instructions have to be managed by external testing applied to the XML metadata. This issue is discussed in more detail in Annex A.

7.3 Special case encodings

7.3.1 Overview of special case encodings

While the encoding mechanisms described in 7.2 cover most classes found in the UML of the ISO 19100 series of International Standards, there are some special cases where different encoding rules are

applied. The exceptions are generally based on special stereotypes applied to classes. A class without a stereotype or a class with the stereotype *Type* or *DataType* will follow the default encodings. Other classes will follow their corresponding encoding rules in 7.3.2 to 7.3.6.

There are some cases of specific XCT encodings described in this section where the XCT is an *xs:simpleType* instead of an *xs:complexType* as described in 7.2. When an XCT is *simple* the corresponding XCPT does not include the "By Ref" capabilities described in 7.2.3.2. Preventing the use of referencing on simple types avoids a potential complexity that would exist if all property types could be implemented by reference.

7.3.2 Abstract classes

7.3.2.1 Overview of abstract class

It is recommended that abstract classes in conceptual models be consistently identified so that implementers realize there cannot be elements of such types.

7.3.2.2 Abstract class rule

Abstract classes are encoded using the default rules, with these additional requirements:

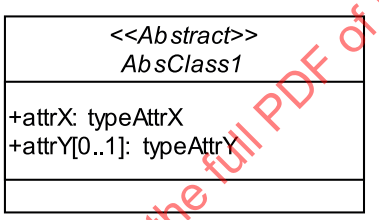

Figure 3 — Sample UML for an abstract class

Requirement /req/abstract/XCT-name

The word 'Abstract' shall be prefixed to the abstract class name of the corresponding XCT.

Requirement /req/abstract/XCT-abstract

The attribute *abstract* shall be added to the *xs:complexType* element of the XCT with a value of "true".

EXAMPLE 1 The XCT corresponding to AbsClass1 as shown in Figure 3:

```
<xs:complexType name="AbstractAbsClass1_Type" abstract="true">
  <xs:complexContent>
    <xs:extension base="gco:AbstractObject_Type">
      <xs:sequence>
        <xs:element name="attrX" type="ns1:typeAttrX_PropertyType"/>
        <xs:element name="attrY" type="ns1:typeAttrY_PropertyType" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Requirement /req/abstract/XCGE-name

The word 'Abstract' shall be prefixed to the abstract class name of the XCGE.

Requirement /req/abstract/XCGE-abstract

The *abstract* attribute with a value of "true" to shall be added the *xs:element* element of the XCGE.

EXAMPLE 2 The XCGE corresponding to AbsClass1 as shown in Figure 3:

```
<xs:element name="AbstractAbsClass1" type="ns1:AbstractAbsClass1_Type" abstract="true"/>
```

Requirement /req/abstract/XCPT-ref

The appropriate prefixed class name shall be used in the *ref* attribute of the *xs:element* in the corresponding XCPT. (Note that the word 'Abstract' is not prefixed to the name attribute of the XCPT because a property type will not be abstract in XML schema).

EXAMPLE 3 The XCPT corresponding to AbsClass1 as shown in Figure 3:

```
<xs:complexType name="AbsClass1_PropertyType">
  <xs:sequence minOccurs="0">
    <xs:element ref="ns1:AbstractAbsClass1"/>
  </xs:sequence>
  <xs:attributeGroup ref="gco:ObjectReference"/>
  <xs:attribute ref="gco:nilReason"/>
 </xs:complexType>
```

7.3.2.3 Abstract class example

Applying the rules above to this UML example (from ISO 19115-1) results in the XML below.

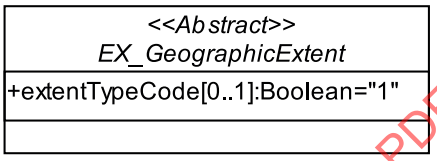

Figure 4 — The UML of the abstract class EX_GeographicExtent from ISO 19115-1

The XCT corresponding to EX_GeographicExtent:

```
<xs:complexType name="AbstractEX_GeographicExtent_Type" abstract="true">
  <xs:complexContent>
    <xs:extension base="gco:AbstractObject_Type">
      <xs:sequence>
        <xs:element name="extentTypeCode" type="gco:Boolean_PropertyType" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

The XCGE corresponding to EX_GeographicExtent:

```
<xs:element name="AbstractEX_GeographicExtent" type="gex:AbstractEX_GeographicExtent_Type"
abstract="true"/>
```

The XCPT corresponding to EX_GeographicExtent:

```
<xs:complexType name="EX_GeographicExtent_PropertyType">
  <xs:sequence minOccurs="0">
    <xs:element ref="gex:AbstractEX_GeographicExtent"/>
  </xs:sequence>
  <xs:attributeGroup ref="gco:ObjectReference"/>
  <xs:attribute ref="gco:nilReason"/>
 </xs:complexType>
```

7.3.3 Inheritance and sub-class encodings

7.3.3.1 Overview of inheritance and sub-class encoding

The concept of inheritance is implemented through generalizations that are considered simple extensions. Admittedly, this is a simplification of the concept of inheritance and sub-classing, since it neglects both cases of generalization relationships that provide restriction and cases of generalization

relationships that address multiple-inheritance. This extension-only characteristic of generalizations is by design. As far as XML schema is concerned, multiple-inheritance is not supported and can only be simulated.

As with the default UML classes, those that are subclasses have corresponding XCT, XCGE and XCPT.

7.3.3.2 Inheritance and sub-class rule

The XCT of a sub-class differs from a default class in the following ways:

Requirement /req/inheritance/XCT

a) The xs:complexContent element of the XCT shall contain an *xs:extension* element whose *base* attribute is equal to the namespace qualified XCT of the base (or super) class. (This differs from the XCT of a default UML class *xs:extension* element whose *base* attribute is a gco:AbstractObject_Type to provide id and uuid attributes. Those necessary attributes still exist through the extension of the base class). It is important to note that if the super class is an abstract class, then its corresponding XCT name will be prefixed with the word "Abstract" as described in 7.3.2.2.

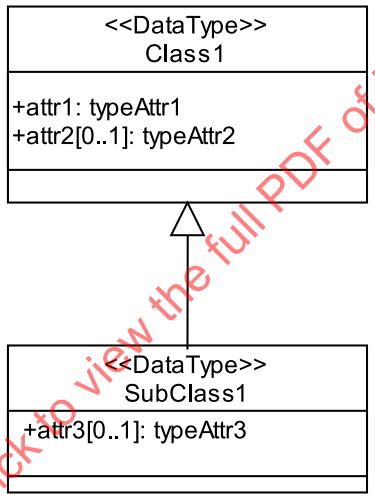

Figure 5 — Sample UML for subclass

EXAMPLE 1 Step a) of building the SubClass1 class shown in Figure 5 is:

```
<xs:complexType name="SubClass1_Type">
    <xs:complexContent>
        <xs:extension base="ns1:Class1_Type">
            (...)
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Requirement /req/inheritance/XCT-sequence

b) The *xs:extension* element shall contain an *xs:sequence* that contains all the properties of the class as described for a default UML Class in /req/default/XCT-properties-sets. Note that this sequence only contains the properties that are specific to the subclass.

EXAMPLE 2 Step a) of building the SubClass1 class shown in Figure 5 is:

```
<xs:complexType name="SubClass1_Type">
  <xs:complexContent>
    <xs:extension base="ns1:Class1_Type">
      <xs:sequence>
        <xs:element name="attr3" type="ns1:typeAttr3_PropertyType"/>
      </xs:sequence>
    </xs:extension>
```

```
  </xs:complexContent>
</xs:complexType>
```

Requirement /req/inheritance/XCGE

The only difference between the XCGE of the subclass and the XCGE of a default UML class is that it shall include the additional *substitutionGroup* attribute, equal to the namespace qualified XCGE of the base class.

EXAMPLE 3 The encoding of the XCGE for the SubClass1 class shown in Figure 5 is:

```
<xs:element name="SubClass1" type="ns1:SubClass1_Type" substitutionGroup="ns1:Class1"/>
```

The XCPT of a subclass is encoded identically to the XCPT of a default UML class. See Requirement /req/default/XCPT and the following.

EXAMPLE 4 The encoding of the XCPT for the SubClass1 class shown in Figure 5 is:

```
xs:complexType name="SubClass1_PropertyType">
    <xs:sequence minOccurs="0">
        <xs:element ref="ns1:SubClass1"/>
    </xs:sequence>
    <xs:attributeGroup ref="gco:ObjectReference"/>
    <xs:attribute ref="gco:nilReason"/>
</xs:complexType>
```

7.3.3.3 Inheritance and sub-class example

EXAMPLE 5 Sample SubClass in XML schema using actual ISO 19115-1 example.

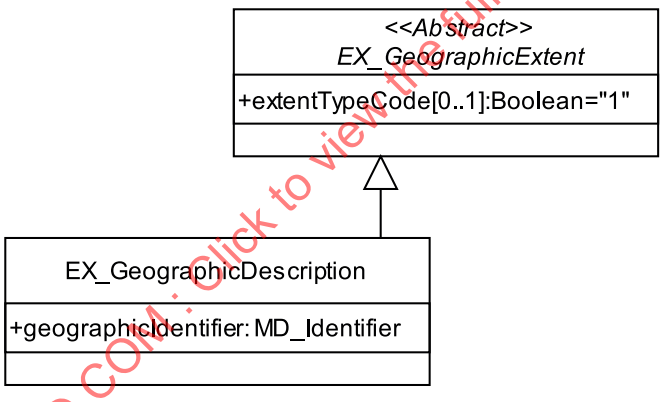

Figure 6 — UML of the classes EX_GeographicExtent and EX_GeographicDescription from ISO 19115-1

EXAMPLE 6 The XCT corresponding to EX_GeographicDescription:

```
<xs:complexType name="EX_GeographicDescription_Type">
    <xs:complexContent>
        <xs:extension base="gex:AbstractEX_GeographicExtent_Type">
            <xs:sequence>
                <xs:element name="geographicIdentifier"
type="mcc:MD_Identifier_PropertyType"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
 </xs:complexType>
```

EXAMPLE 7 The XCGE corresponding to EX_GeographicDescription:

```
<xs:element name="EX_GeographicDescription" type="gex:EX_GeographicDescription_Type" subst
itutionGroup="gex:AbstractEX_GeographicExtent"/>
```

EXAMPLE 8 The XCPT corresponding to EX_GeographicDescription:


```
<xs:complexType name="EX_GeographicDescription_PropertyType">
    <xs:sequence minOccurs="0">
        <xs:element ref="gex:EX_GeographicDescription"/>
    </xs:sequence>
    <xs:attributeGroup ref="gco:ObjectReference"/>
    <xs:attribute ref="gco:nilReason"/>
</xs:complexType>
```

7.3.4 Enumeration encodings

7.3.4.1 Overview of enumeration encodings

The conceptual schema language of the ISO 19100 series of International Standards supports two enumerated types whose "declaration defines a list of valid mnemonic identifiers (ISO 19103)". The two enumerated types are *Enumeration* and *CodeList*, and it is important to recognize the difference between these two types.

A class with the keyword *enumeration* "is a fixed list of valid identifiers of named literal values. Attributes of an enumerated type may only take values from this list." (ISO 19103).

An <<enumeration>> class is used only in the case where no extension to the list of values is necessary; otherwise a <<CodeList>> class is used.

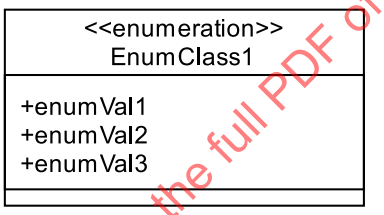

Figure 7 — Sample UML for class stereotyped enumeration

7.3.4.2 Enumeration rule

Requirement /req/enumeration/XCT

The encoding of an <<enumeration>> class shall begin with the creation of an XCT starting with an *xs: simpleType* element, with a *name* attribute equal to the class name in the UML and with the addition of the _Type suffix.

EXAMPLE 1 Step a) of building the EnumClass1 class (shown in Figure 7) is:

```
<xs:simpleType name="EnumClass1_Type">
        (...)
</xs:simpleType>
```

Requirement /req/enumeration/XCT-string

The xs:simpleType element shall contain an xs:restriction element, with a base attribute equal to an xs:string.

EXAMPLE 2 Step b) of building the EnumClass1 class shown in Figure 7 is

```
<xs:simpleType name="EnumClass1_Type">
  <xs:restriction base="xs:string">
      (...)
  </xs:restriction>
</xs:simpleType>
```

Requirement /req/enumeration/XCT-enumerations

Each *xs:restriction* element shall contain a series of *xs:enumeration* elements whose value attributes equal the attribute values in the UML model for the enumeration.

EXAMPLE 3 Step c) of building the EnumClass1 class shown in Figure 7 is:

```
<xs:simpleType name="EnumClass1_Type">
    <xs:restriction base="xs:string">
        <xs:enumeration value="enumVal1"/>
        <xs:enumeration value="enumVal2"/>
        <xs:enumeration value="enumVal3"/>
    </xs:restriction>
</xs:simpleType>
```

Requirement /req/enumeration/XCGE

The XCGE of an <<enumeration>> class shall follow the default XCGE rules (/req/default/XCGE), with the addition of a *substitutionGroup* attribute equal to the "gco:CharacterString" XCGE.

EXAMPLE 4 The XCGE of the EnumClass1 class shown in Figure 7 is:

```
<xs:element name="EnumClass1" type="ns1:EnumClass1_Type" substitutionGroup="gco:CharacterS
tring"/>
```

Requirement /req/enumeration/XCPT

The XCPT of an <<enumeration>> class shall follow the default encoding described for simple type XCTs in 7.3.1.

EXAMPLE 5 The XCPT of the EnumClass1 class shown in Figure 7 is:

```
<xs:complexType name="EnumClass1_PropertyType">
    <xs:sequence minOccurs="0">
        <xs:element ref="ns1:EnumClass1"/>
    </xs:sequence>
    <xs:attribute ref="gco:nilReason"/>
</xs:complexType>
```

7.3.4.3 Enumeration example

Applying the enumeration encoding rule above to this UML example (from ISO 19115-1) results in the XML below.

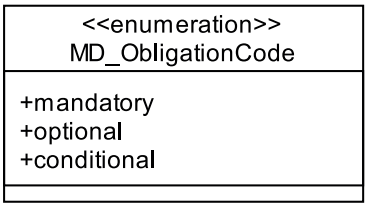

Figure 8 — MD_ObligationCode class from ISO 19115-1

EXAMPLE 6 The XCT corresponding to MD_ObligationCode:

```
<xs:simpleType name="MD_ObligationCode_Type">
    <xs:restriction base="xs:string">
        <xs:enumeration value="mandatory"/>
        <xs:enumeration value="optional"/>
        <xs:enumeration value="conditional"/>
    </xs:restriction>
</xs:simpleType>
```

EXAMPLE 7 The XCGE corresponding to MD_ObligationCode:


```
<xs:element name="MD_ObligationCode" type="mex:MD_ObligationCode_Type" substitutionGroup="
gco:CharacterString"/>
```

EXAMPLE 8 The XCPT corresponding to MD_ObligationCode:

```
<xs:complexType name="MD_ObligationCode_PropertyType">
    <xs:sequence minOccurs="0">
        <xs:element ref="mex:MD_ObligationCode"/>
    </xs:sequence>
    <xs:attribute ref="gco:nilReason"/>
</xs:complexType>
```

7.3.5 CodeList encoding

7.3.5.1 Overview and building blocks of the CodeList encoding

A class that is stereotyped CodeList is an enumerated type like a class with the keyword *enumeration*. The difference is that the <<CodeList>> class is extensible.

<<CodeList>> classes are sometimes described as part of the data dictionary, in tables with two columns: Name and Definition.

Table 2 — CI_TelephoneTypeCode <<CodeList>> table from ISO 19115-1

Name	Definition
CI_TelephoneTypeCode	type of telephone
voice	telephone provides voice service
fax	telephone provides facsimile service
sms	telephone provides sms service

Each row in a CodeList table is intended to describe a unique concept or option that supports the intent of the CodeList's definition. The specific encoding of the CodeList stereotype is meant to support some characteristics intended by the creators. These characteristics are:

a) CodeLists and their associated definitions are controlled in registers. The data dictionary is the base concept register of the CodeLists, and is the source for the establishment of registers for user communities.

b) The Name column of the CodeList tables contains values that all software will use to recognize each unique concept or option defined in a row of a CodeList table. Consequently, the 'Name' value should not be regarded as a proper name (in any language, including English).

c) User communities may need to add new concepts or options that were not considered by the creators of the codelist, and it is necessary to control these extensions in user community registers. (This would be like adding additional rows to a CodeList in the data dictionary).

7.3.5.2 CodeList encoding rule

Requirement /req/codelist/XCT

To satisfy the intended characteristics of a CodeList, a special "CodeListValue_Type" XCT is provided in the gco namespace, with two attributes, as shown in the fragment below:

```
<xs:complexType name="CodeListValue_Type">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute name="codeList" type="xs:anyURI"  use="required"/>
            <xs:attribute name="codeListValue" type="xs:anyURI"
use="required"/>
        </xs:extension>
    </xs:simpleContent>
```

```
</xs:complexType>
```

The definition of the special XCT makes any CodeList substitutable to a CharacterString.

An XCGE based on this special XCT and an XCPT is defined for each CodeList as described in 7.2.2. Consequently, any CodeList instance is an XML element. Its name is the name of its XCGE. It contains a text value and the two XML attributes defined by the special XCT.

The *codeList* attribute is intended to contain a URL that references a codeList's details, within a registry or a codelist catalogue.

EXAMPLE 1 If Table 2 was contained in a catalogue of CodeLists on the ISO web site, then a possible value for the codeList attribute might be:

```
codeList = "http://standards.iso.org/iso/19115/-1/cit/1.0/codelists.xml#CI_
TelephoneTypeCode"
```

The codeListValue attribute is intended to carry the identifier of the codelist value's definition. This identifier is the value expressed in the name column of data dictionary tables. The codelist catalogue (or registry) is expected to contain an explicit name and definition of the value in the default language of the metadata, as well as alternate expressions in different code spaces, some of them corresponding to the different locales supported by the geographic resource.

EXAMPLE 2 If a metadata instance document were intended to indicate a telephone type of "voice", then the value (within the metadata) for the codeListValue (based on Table 2), would be:

```
codeListValue="voice"
```

The content of the element is the name of the codelist value in the default language of the metadata.

EXAMPLE 3 Below is a more complete metadata implementation instance excerpt, using a codelist value from the CI_TelephoneTypeCode:

```
<CI_TelephoneTypeCode codeList="http://standards.iso.org/iso/19115/-1/cit/1.0/
codelists.xml#CI_TelephoneTypeCode" codeListValue="voice">voice</CI_TelephoneTypeCode>
```

The XML element value allows the user to access a valid default expression of the actual value of the CodeList, while the two attributes allow an application to access the full definition of the CodeList and its values, typically for customization of the user interface (creation of a drop-down box providing the registered or catalogued value of the codelist, multilingual and multicultural management, ...)

7.3.5.3 Details of the CodeList encoding

The CodeListValue_Type was introduced in 7.3.5.2, and that complexType serves as the XCT for all <<CodeList>> classes.

Requirement /req/codelist/XCGE

The XCGE of a <<CodeList>> class shall be encoded as a global element with a *name* attribute equal to the name of the UML class and the *type* equal to the CodeListValue_Type. Like an <<enumeration>> class, this XCGE shall also contain a *substitutionGroup* attribute equal to the "gco:CharacterString" XCGE.

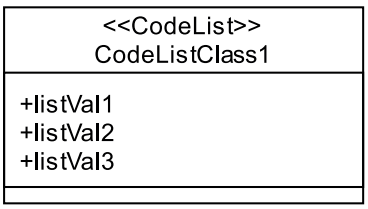

Figure 9 — Sample UML for class stereotyped CodeList

EXAMPLE 1 The XCGE defined for the CodeListClass1 Class shown in Figure 9 is:

```
<xs:element name="CodeListClass1" type="gco:CodeListValue_Type"
substitutionGroup="gco:CharacterString"/>
```

The XCPT for a <<CodeList>> class follows the encoding rules defined in 7.3.1 for simple type encodings.

EXAMPLE 2 The XCPT defined for the CodeListClass1 Class shown in Figure 9 is:

```
<xs:complexType name="CodeListClass1_PropertyType">
   <xs:sequence minOccurs="0">
        <xs:element ref="ns1:CodeListClass1"/>
   </xs:sequence>
   <xs:attribute ref="gco:nilReason"/>
</xs:complexType>
```

EXAMPLE 3 Sample CodeList Class encoding using actual ISO 19115-1 example.

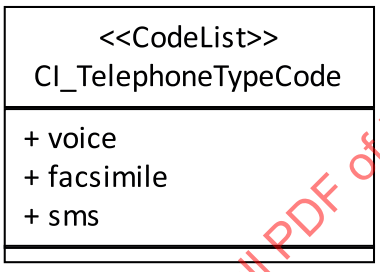

Figure 10 — CI_TelephoneTypeCode Class from ISO 19115-1

EXAMPLE 4 The XCGE corresponding to CI_DateTypeCode:

```
<xs:element name="CI_TelephoneTypeCode" type="gco:CodeListValue_Type"
substitutionGroup="gco:CharacterString"/>
```

EXAMPLE 5 The XCPT corresponding to CI_TelephoneTypeCode:

```
<xs:complexType name="CI_TelephoneTypeCode_PropertyType">
      <xs:sequence minOccurs="0">
          <xs:element ref="cit:CI_TelephoneTypeCode"/>
      </xs:sequence>
      <xs:attribute ref="gco:nilReason"/>
   </xs:complexType>
```

7.3.6 Union encoding

7.3.6.1 Overview of the Union encoding rule

The Union encoding rule offers a choice between several items.

7.3.6.2 Union encoding rule

Requirement /req/union/XCT

All classes with the stereotype *Union,* like that shown in Figure 11, shall be treated as follows:

— the class will be transformed into an XCT via an *xs:complexType,* with the name attribute equal to the class name in the UML and with the addition of the _Type suffix.

Within the *xs:complexType* element there will be an *xs:choice* element that contains one *xs:element* for each attribute of the UML class.

For each xs:element :

— the *name* attribute will be equal to the name of the attribute in the UML class; and

— the *type* attribute will be equal to the type of the attribute, but prefixed with the proper namespace and suffixed with "_PropertyType".

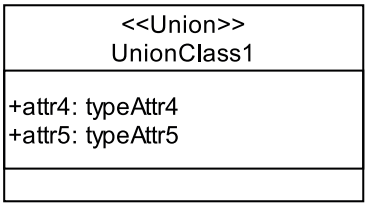

Figure 11 — Sample UML for class stereotyped Union

EXAMPLE 1 The XCT defined for the UnionClass1 class shown in Figure 11 is:

```
<xs:complexType name="UnionClass1_Type">
    <xs:choice>
        <xs:element name="attr4" type="ns1:typeAttr4_PropertyType"/>
        <xs:element name="attr5" type="ns1:typeAttr5_PropertyType"/>
    </xs:choice>
</xs:complexType>
```

Requirement /req/union/XCGE

The XCGE of a <<Union>> class shall follow the default encoding rules described in 7.2.

EXAMPLE 2 The XCGE defined for the UnionClass1 class shown in Figure 11 is:

```
<xs:element name="UnionClass1" type="ns1:UnionClass1_Type"/>
```

Requirement /req/union/XCPT

Since a Union is another case where referencing is undesirable, the XCPT shall follow the default encoding rules for simple types as described in 7.3.1.

EXAMPLE 3 The XCPT defined for the UnionClass1 class shown in Figure 11 is:

```
xs:complexType name="UnionClass1_PropertyType">
    <xs:sequence minOccurs="0">
      <xs:element ref="ns1:UnionClass1"/>
    </xs:sequence>
    <xs:attribute ref="gco:nilReason"/>
</xs:complexType>
```

7.3.6.3 Union encoding example

Applying the Union encoding rule above to the UML Union example in Figure 12 (from ISO 19115-1) results in the XML below.

Figure 12 — The MD_Resolution class from ISO 19115-1

EXAMPLE 5 The XCT corresponding to MD_Resolution:

```
<xs:complexType name="MD_Resolution_Type">
    <xs:choice>
        <xs:element name="equivalentScale"
             type="mri:MD_RepresentativeFraction_PropertyType"/>
        <xs:element name="distance" type="gco:Distance_PropertyType"/>
        <xs:element name="vertical" type="gco:Distance_PropertyType"/>
        <xs:element name="angularDistance" type="gco:Angle_PropertyType"/>
        <xs:element name="levelOdDetail" type="gco:CharacterString_PropertyType"/>
    </xs:choice>
</xs:complexType>
```

EXAMPLE 6 The XCGE corresponding to MD_Resolution:

```
<xs:element name="MD_Resolution" type="mri:MD_Resolution_Type"/>
```

EXAMPLE 7 The XCPT corresponding to MD_Resolution:

```
<xs:complexType name="MD_Resolution_PropertyType">
      <xs:sequence minOccurs="0">
          <xs:element ref="mri:MD_Resolution"/>
      </xs:sequence>
      <xs:attribute ref="gco:nilReason"/>
</xs:complexType>
```

7.3.7 Encoding of MetaClasses

MetaClasses can only be instantiated through one of their realizations. The encoding and use of realizations is discussed in greater detail below.

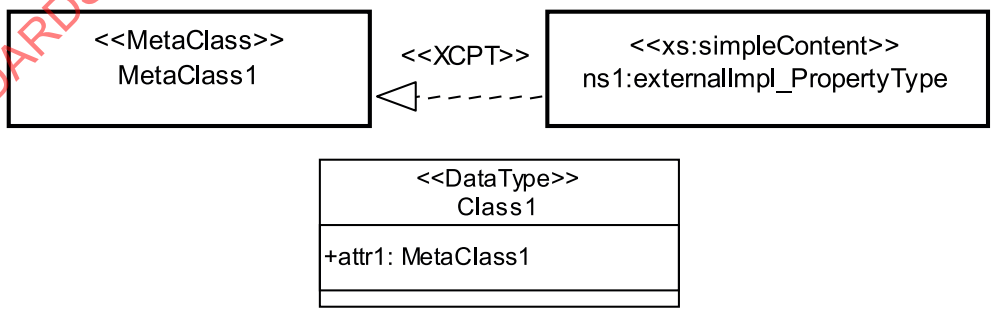

Figure 13 — Sample UML for class stereotyped MetaClass

The realization of a <<MetaClass>> is used when determining the type attribute for an element, in the sequence of a default encoded XCT. In other words, in the XCT of Class1 from Figure 13, instead of the type for the "attr1" element being a "MetaClass1_PropertyType", it would be a "ns1:externalImpl

_PropertyType", because of the realization shown between MetaClass1 and ns1:externalImpl _PropertyType.

EXAMPLE The XCT corresponding to the Class1 class shown in Figure 13 is:

```
<xs:complexType name="Class1_Type">
    <xs:complexContent>
        <xs:extension base="gco:AbstractObject_Type">
            <xs:sequence>
                <xs:element name="attr1" type="ns1:externalImpl_PropertyType"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

7.3.8 Encoding of externally identified implementations

7.3.8.1 Overview of externally identified implementations

The use of externally identified implementations was introduced in 6.5, to improve the interoperability and usability of this document. It is possible to use external implementations to take advantage of implementations of the ISO 19100 series of International Standards that already exist, or to use external implementations that are specific to a particular encoding technology. The UML Notation for *realization* is used to indicate, in UML, where the XML schemas defined in other ISO 19100 International Standards are being used.

7.3.8.2 Encoding options

There are three ways of encapsulating external implementations into concepts depicted in the ISO 19100 series of UML models. The names from the ISO 19100 series UML classes are preserved as an entry point into the implementation schema based on the stereotype of the *realization* as described in ISO 19103.

7.3.8.2.1 Encoding through XCPT

Requirement /req/external/XCPT

The simplest case of using an external encoding occurs when the existing implementation provides class types, global elements and class property types for the specified ISO 19100 series UML classes that match the default encoding rules described in 7.2. This is indicated in UML by the existence of an XCPT stereotype on a realization relationship, as shown in Figure 14.

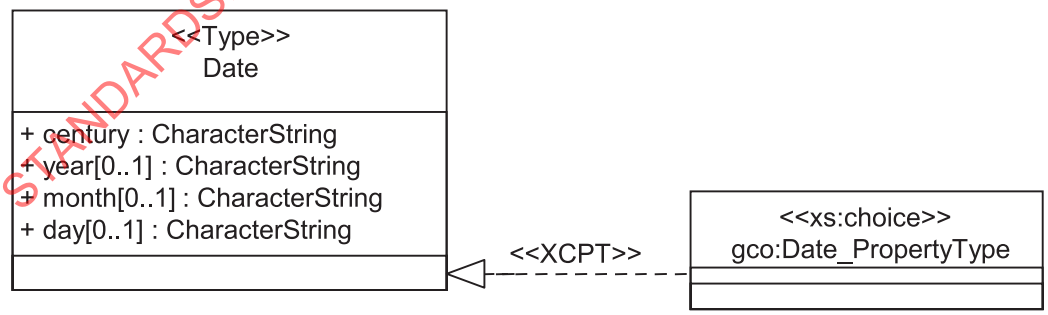

Figure 14 — ISO 19103 date realized by gco:Date_PropertyType

In this situation, no XCTs, XCGEs or XCPTs shall be created for the target class. When the target class is the type for a property of an XCT, then the name of the source class shall be used as the value for the type attribute. In most cases the value for the type attribute will follow the default encoding rule that uses the target class name + "_PropertyType", but this is not always the situation.

Figure 15 shows the UML for the CI_Date class.

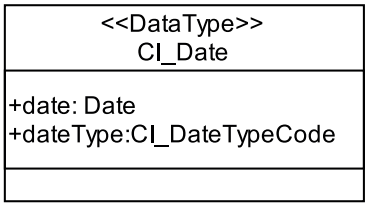

Figure 15 — CI_Date class from ISO 19115-1

EXAMPLE The XCT corresponding to CI_Date shown in Figure 15 is:

```
<xs:complexType name="CI_Date_Type">
    <xs:complexContent>
        <xs:extension base="gco:AbstractObject_Type">
            <xs:sequence>
                <xs:element name="date" type="gco:Date_PropertyType"/>
                <xs:element name="dateType" type="cit:CI_DateTypeCode_PropertyType"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

7.3.8.2.2 Encoding through XCGE

Requirement /req/external/XCGE

The next case of using an external encoding occurs when the existing implementation provides class types and global elements that can be used, but where the inheritance trees of the external implementation should be preserved.

In this case, the following rules should be applied:

a) an XCPT will be created using the class name from the ISO 19100 series UML models. This XCPT serves as an entry point into the implementation schema, through the external implementation's XCGE. This is indicated in UML by the existence of an XCGE stereotype on a realization relationship, as shown in Figure 16 below.

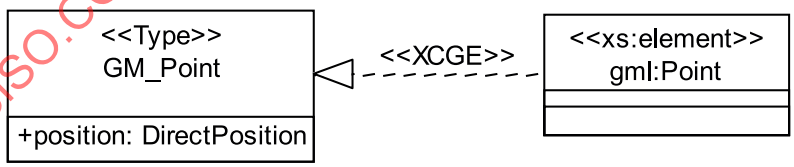

Figure 16 — ISO 19107 GM_Point realized by gml:Point

b) When the realization has an XCGE stereotype, the XCPT for the target class will follow the encoding rules described in 7.4 for the encoding of the XCPT and use the name of the realization's source class as the value of the ref attribute, in the xs:element tag.

EXAMPLE 1 The XCPT for GM_Point based on Figure 16:

```
<xs:complexType name="GM_Point_PropertyType">
    <xs:sequence minOccurs="0">
        <xs:element ref="gml:Point"/>
    </xs:sequence>
    <xs:attributeGroup ref="gco:ObjectReference"/>
    <xs:attribute ref="gco:nilReason"/>
</xs:complexType>
```

Requirement /req/external/XCGE-element

In certain cases, the UML representing the external implementation may contain properties of the class stereotyped xs:element. If this is the case then there will be an XCGE corresponding to the class shown in the UML that resides in the namespace indicated by the namespace prefix used in the class name. That XCGE shall be defined using an xs:element with the name attribute equal to the name of the XML class shown in the UML minus the namespace prefix. The *type* attribute shall be equal to the only named property shown in the UML. Any other attributes of the xs:element shall be indicated as properties of the XML class shown in UML as the type of the property, and having a value sent to any default value displayed in the UML.

EXAMPLE 2 Sample XCGE based on an xs:element in the gco namespace.

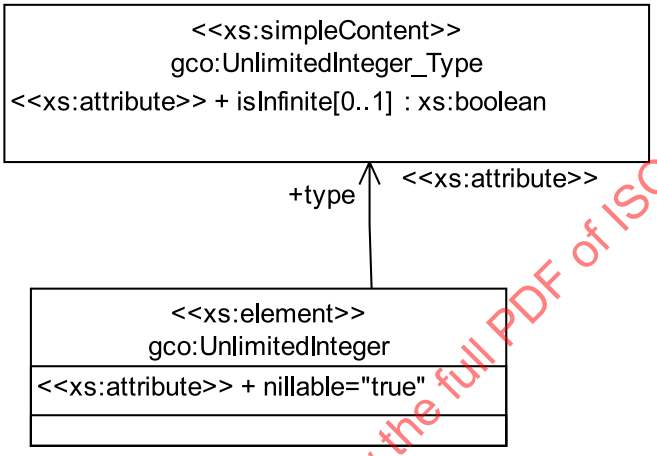

Figure 17 — XCGE for gco:UnlimitedInteger shown as xs:element stereotyped class

EXAMPLE 3 The XCGE corresponding to Unlimited Integer:

```
<xs:element name="UnlimitedInteger" type="gco:UnlimitedInteger_Type" nillable="true"/>
```

7.3.8.2.3 Encoding through XCT

7.3.8.2.3.1 General

Requirement /req/external/XCT-simpleType

Often, only an XCT is provided in the external implementation. This is the case when XML schema *simpleTypes* are utilized. When this is the situation, the realization from the ISO 19100 series class to the XML schema object carries an XCT stereotype.

7.3.8.2.3.2 The xs:simpleType stereotype

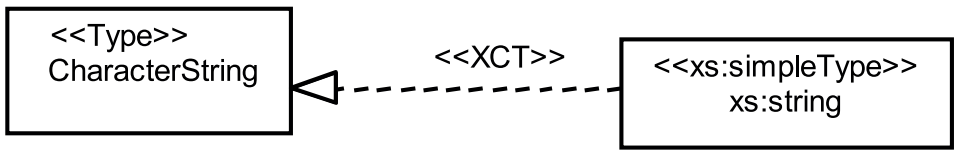

Figure 18 — Realization of CharacterString with xs:string

When the realization has an XCT stereotype and the source class has an *xs:simpleType* stereotype the XCGE shall follow the encoding rules described in 7.3 for the encoding of the XCGE and use the name of the corresponding XCT as the value of the type attribute.

EXAMPLE The XCGE for CharacterString based on Figure 18:

```
<xs:element name="CharacterString" type="xs:string"/>
```

7.3.8.2.3.3 The xs:simpleContent stereotype

Requirement /req/external/XCT-simpleContent

When the realization has an XCT stereotype and the source class has an *xs:simpleContent* stereotype an XCT shall be created corresponding to the source class.

The XCT shall conform to these requirements:

a) an xs:complexType will be created with a *name* attribute equal to the name of the xml class minus any namespace prefix. If the xml class happens to be abstract, then the xs:complexType will contain an *abstract* attribute with the value set to “true”;

b) The xs:complexType will contain an xs:simpleContent;

c) The xs:simpleContent will contain an xs:extension with a *base* attribute set to the name of the external implementation super class;

d) The xs:simpleContent will contain occurrences of xs:attribute or xs:attributeGroup depending on the properties and their respective stereotypes shown in the UML representing the XML class.

   If there is a type value present in the UML property, then the xs:attribute or xs:attributeGroup shall have:

   1) a name attribute equal to the name of the property in the UML, and

   2) a *type* attribute equal to the type shown in the UML.

   If there is no type present for a property in the UML, then the xs:attribute or xs:attributeGroup shall contain a ref attribute with a value equal to the name of the property in the UML.

   Just as with default XCT encodings from 7.2, the minOccurs and maxOccurs values are based on ISO 19118:2011, Table A.5.

EXAMPLE Sample XCT realization on an xs:simpleConten,t in the gco namespace.

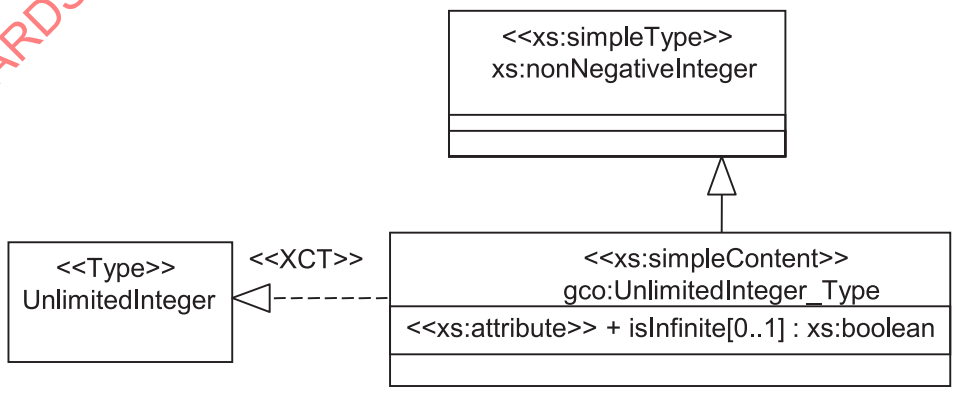

Figure 19 — XCT realize for UnlimitedInteger_Type with xs:simpleContent stereotype

The XCT corresponding to UnlimitedInteger:

```
<xs:complexType name="UnlimitedInteger_Type">
     <xs:simpleContent>
          <xs:extension base="xs:nonNegativeInteger">
               <xs:attribute name="isInfinite" type="xs:boolean"/>
          </xs:extension>
     </xs:simpleContent>
</xs:complexType>
```

7.3.8.2.3.4 The xs:complexType stereotype

Requirement /req/external/XCT-complexType

When the realization has an XCT stereotype and the source class has an *xs:complexType* stereotype, an XCT shall be created corresponding to the source class.

The XCT shall conform to these requirements:

a) an xs:complexType will be created with a *name* attribute equal to the name of the XML class minus any namespace prefix. If the XML class happens to be abstract, then the xs:complexType shall contain an *abstract* attribute with the value set to "true";

b) The xs:complexType will contain an xs:complexContent;

c) The xs:complexContent will contain an xs:extension, with a *base* attribute set to the name of the external implementation super class.

   If there is no external implementation super class, then the base attribute will be set to gco: AbstractObject;

d) The xs:extension will contain an xs:sequence composed of xs:elements based on the properties of the XML class which are stereotyped xs:element.

   Any property with the xs:element stereotype indicates the presence of an xs:element in the xs: sequence with:

   1) a *name* attribute equal to the name of the property, and

   2) the *type* attribute equal to the property type.

   Just as with default XCT encodings from 7.2, the minOccurs and maxOccurs values are based on ISO 19118:2011, Table A.5;

e) After the xs:sequence element, there may also be occurences of xs:attribute or xs:attributeGroup based on any properties of the XML class that are stereotyped xs:attribute or xs:attributeGroup. If there is a type value present in the UML element, then the xs:attribute or xs:attributeGroup will have:

   1) a *name* attribute equal to the name of the element in the UML, and

   2) a type attribute equal to the type shown in the UML.

   If there is no type present for an element in the UML, then the xs:attribute or xs:attributeGroup shall contain a *ref* attribute, with a value equal to the name of the element in the UML.

   Just as with default XCT encodings from 7.2, the minOccurs and maxOccurs values are based on ISO 19118:2011, Table A.5.

EXAMPLE Sample XCT realization on an xs:complexType in the lan namespace.

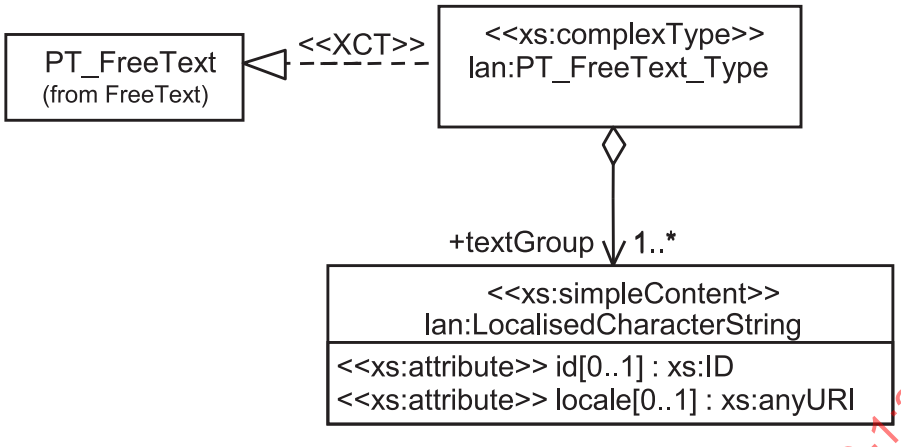

Figure 20 — XCT realize for gmd:PT_FreeText_Type with xs:complexType stereotype

The XCT corresponding to PT_FreeText:

```
<xs:complexType name="PT_FreeText_Type">
    <xs:complexContent>
       <xs:extension base="gco:AbstractObject_Type">
         <xs:sequence>
                <xs:element name="textGroup"
type="lan:LocalisedCharacterString_PropertyType"
                  maxOccurs="unbounded"/>
         </xs:sequence>
       </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

7.3.8.2.3.5 The xs:union stereotype

Requirement /req/external/XCT-union

When a class has an xs:union stereotype, the following rules shall be applied:

a) an XCT will be created corresponding to the UML class.

b) The XCT will be an xs:simpleType with the *name* attribute equal to the name of the UML class.

c) Within the xs:simpleType, there will be one xs:union with a *member* attribute equal to the names of the elements in the UML class, separated by spaces.

EXAMPLE Sample XCT for an xs:union in the gco namespace.

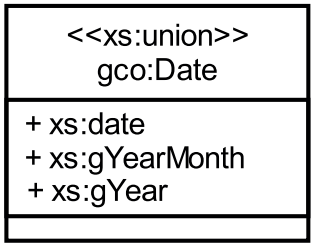

Figure 21 — gco:Date XCT with stereotype xs:union

The XCT corresponding to Date:

```
<xs:simpleType name="Date_Type">
    <xs:union memberTypes="xs:date xs:gYearMonth xs:gYear"/>
```

```
</xs:simpleType>
```

7.3.8.2.4 Creating the XCGE when encoding through the XCT

Requirement /req/external/XCGE

An XCGE shall be created for any external implementation XCTs. The XCGEs follow the default encoding rules described in 7.2 and 7.3.2 (for the case of abstract classes) and 7.3.3 (for the case of subclasses).

7.3.8.2.5 Creating the XCPT

Requirement /req/external/XCPT

An XCPT shall also be created following the encoding rules in 7.2.3 if

a) the source class of the realization is stereotyped *xs:complexType*, or

b) (following the encoding rules in 7.3.1), the source class of the realization is stereotyped *xs:simpleType* or *xs:simpleContent*.

EXAMPLE 1 The XCPT for CharacterString based on Figure 32 is:

```
<xs:complexType name="CharacterString_PropertyType">
    <xs:sequence minOccurs="0">
        <xs:element ref="gco:CharacterString"/>
    </xs:sequence>
    <xs:attribute ref="gco:nilReason"/>
</xs:complexType>
```

If a class is stereotyped *xs:choice,* then the XCPT will be slightly modified from the default encoding rules.

It is assumed that a class stereotyped *xs:choice* represents a unique XCP, and as such, its name should already include the suffix, "_PropertyType". This name will be used as the name attribute in the xs:complexType element for the XCPT, which will also contain an *xs:choice* with a minOccurs attribute set to "0".

Within the xs:choice will be an *xs:element* corresponding to each property of the UML class. Each xs:element shall contain one ref attribute containing the name of the XCGE specified as the property of the UML class. After the *xs:choice* is closed, there is one gco:nilReason, and then the XCPT is closed.

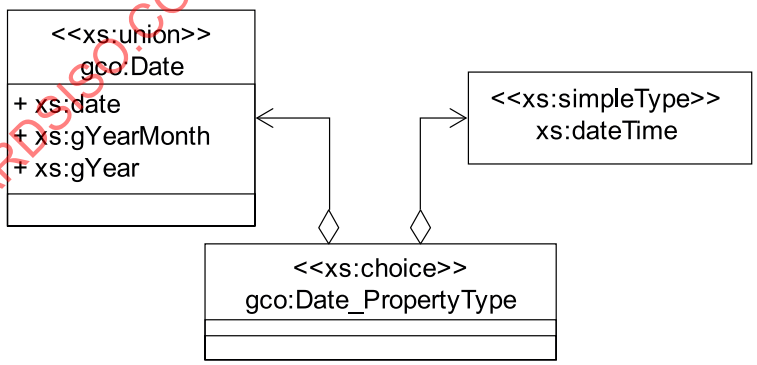

Figure 22 — Sample UML for class stereotyped xs:choice

EXAMPLE 2 The XCPT for gco:Date_PropertyType based on Figure 22 is:

```
<xs:complexType name="Date_PropertyType">
  <xs:choice minOccurs="0">
        <xs:element ref="gco:Date"/>
        <xs:element ref="gco:DateTime"/>
     </xs:choice>
     <xs:attribute ref="gco:nilReason"/>
```

```
</xs:complexType>
```

7.4 XML Namespace package encoding

The stereotype *xmlNamespace* was introduced in 4.3 as a set of XML objects grouped within the same namespace. A UML package can have dependencies that are defined as a relationship between two modelling elements, in which a change to one modelling element will affect the other modelling element (ISO 19103).

In the case of an <<xmlNamespace>> package, there are two types of dependency relationships to consider.

a) The stereotype *implement* on a dependency has a source representing the XML namespace (or namespace prefix) that implements the abstract concepts defined in the target. An introduction to the mrl namespace is provided in ISO/TS 19115-3, and Figure 23 shows the dependency in UML, using an <<xmlNamespace>> package and an <<implement>> dependency.

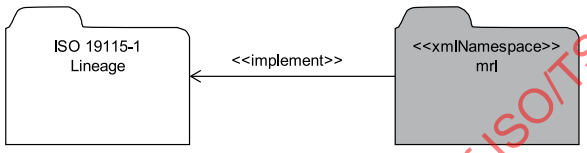

Figure 23 — xmlNamespace and implement

b) The stereotype *import* on a dependency has a source representing objects grouped within one namespace that depend on objects grouped within another namespace. An <<import>> dependency will exist between two <<xmlNamespace>> packages if there is a dependency relationship between the two abstract packages that are the target of <<xmlNamespace>> package <<implement>> dependencies.

   In Figure 24 an <<import>> dependency is shown between the mri package and the gco package, which are both defined in ISO/TS 19115-3. The <<import>> dependency is used to reference an XML schema file from a different namespace (mri and gco in this example).

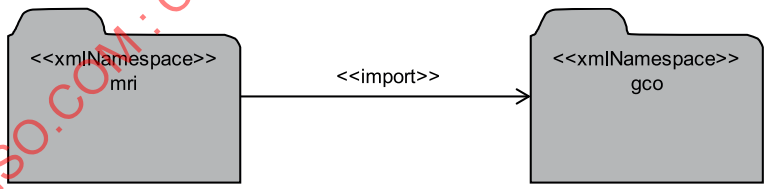

Figure 24 — xmlNamespace and import

The <<xmlNamespace>> package is not directly encoded into XML but it will have a corresponding <<xmlSchema>> package that utilizes the <<implement>> and <<import>> dependency relationships to generate appropriate XML schema.

7.5 XML schema package encoding

A package stereotyped xmlSchema, as noted in 4.3, is a package that represents an XML schema. For any <<xmlSchema>> package in the UML there shall exist an XML schema file with the same name as the package. An <<xmlSchema>> package can utilize an <<implement>> dependency exactly as described in 7.4 for an <<xmlNamespace>> and it can also utilize an <<include>> dependency. The <<include>> dependency is used to reference an XML schema file from the same namespace. Each <<xmlSchema>> package is one of two types.

The first type is a root <<xmlSchema>> package that corresponds directly to an <<xmlNamespace>> package and has the same name as the <<xmlNamespace>> package with the addition of the .xsd

extension. This means that because there is an mri <<xmlNamespace>> package shown in Figure 24 there will also be a corresponding mri.xsd package with the stereotype *xmlSchema* and accordingly a mri.xsd file that is created using the encoding rules defined in this document.

Each XML schema file starts with this XML declaration: *<?xml version="1.0" encoding="utf-8"?>*. The root element in the same file is an xs:schema and shall have the attributes shown in Table 3.

Table 3 — Attributes of xs:schema element for a root <<xmlSchema>> package

Occur-rences of attribute	Name-space of attribute	Name of attribute	Value of attribute	Example attribute-value pair
One	N/A	targetNamespace	http://standards.iso.org/iso/<-standardnumber>/<part>/*<packageName>*/<namespace version number>	targetNamespace="http://standards.iso.org/iso/19115/-3/mri/1.0/"
One	xmlns	*xmlns*	http://standards.iso.org/iso/<-standardnumber>/<part>/*<packageName>*/<namespace version number>	xmlns:mri="http://standards.iso.org/iso/19115/-3/mri/1.0/"
Zero to many	xmlns	*packageName of target of <<import>> dependency*	http://standards.iso.org/iso/<-standardnumber>/<part>/*<packageName>*/<namespace version number> *of target of <<import>> dependency*	xmlns:mri="http://standards.iso.org/iso/19115/-3/mri/1.0/"
One	xmlns	xs	http://www.w3.org/2001/XMLSchema	xmlns:xs="http://www.w3.org/2001/XMLSchema"
One	N/A	version	<schema version number>	version="1.2.3"

NOTE 1 *packageName* in Table 3 refers to the name of the corresponding <<xmlNamespace>> package of a root <<xmlSchema>> package.

EXAMPLE The UML shown in Figure 25 would result in the existence of a file named mri.xsd with the following declaration of xs:schema:

```
<xs:schema targetNamespace=http://standards.iso.org/iso/19115/-3/mri/1.0
xmlns:lan="http://standards.iso.org/iso/19115/-3/lan/1.0"
xmlns:mcc=http://standards.iso.org/iso/19115/-3/mcc/1.0
xmlns:mri="http://standards.iso.org/iso/19115/-3/mri/1.0" version="1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

The second type of <<xmlSchema>> package is referred to as a regular <<xmlSchema>> package and it corresponds directly to an existing package from the ISO 19100 series which is indicated by a dependency with the stereotype *implement*. The example in Figure 25 shows a regular <<xmlSchema>> package and its target ISO 19115-1 package.

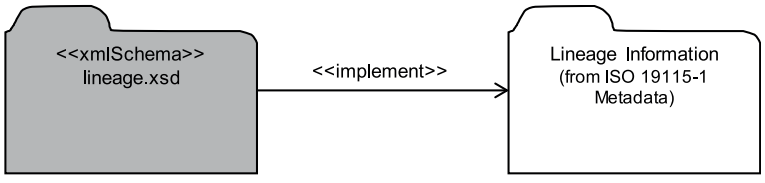

Figure 25 — Regular xmlSchema package

All regular <<xmlSchema>> packages have exactly one corresponding root <<xmlSchema>> package which is either the direct source of an <<include>> dependency between the regular <<xmlSchema>> package or is the root <<xmlSchema>> of a regular <<xmlSchema>> that is the direct source of an <<include>> dependency to the regular <<xmlSchema>> of interest. In Figure 26 the root

<<xmlSchema>> is root.xsd and the regular <<xmlSchema>> are schema1.xsd, and schema2.xsd. The root.xsd <<xmlSchema>> is the root of the two regular <<xmlSchema>>.

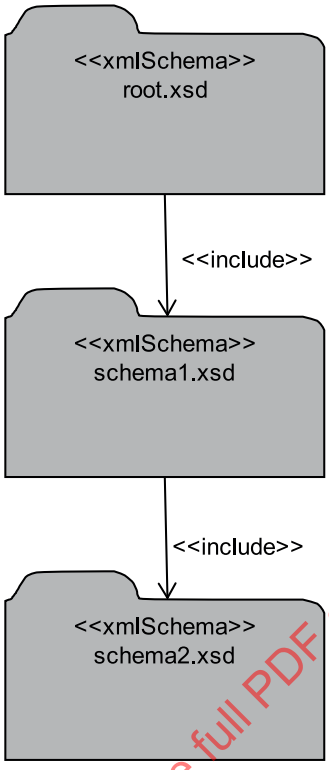

Figure 26 — Hierarchy of <<xmlSchema>> packages

The XML declaration within each XML schema file that is created based on the existence of a regular <<xmlSchema>> package is *<?xml version="1.0" encoding="utf-8"?>*. The root element in the same file is an xs:schema with the attributes shown in Table 4.

Table 4 — Attributes of xs:schema element for a regular <<xmlSchema>> package

Occurances of attribute	Namespace of attribute	Name of attribute	Value of attribute	Example attribute-value pair
One	N/A	targetNamespace	http://standards.iso.org/iso/<standardnumber>/<part>/<rootPackagename>/<namespace version number>	targetNamespace="http://standards.iso.org/iso/19115/-3/mri/1.0/"
One	xmlns	*rootPackageName*	http://standards.iso.org/iso/<standardnumber>/<part>/*<rootPackagename>*/<namespace version number>	xmlns:mri=" http://standards.iso.org/iso/19115/-3/mri/1.0/"
Zero to many	xmlns	*rootPackageName of target of <<import>> dependency*	http://standards.iso.org/iso/<standardnumber>/<part>/*<rootPackageName of target of <<import>> dependency>*/<namespace version number>	xmlns:mri=" http://standards.iso.org/iso/19115/-3/mri/1.0/"

Table 4 *(continued)*

Occurrences of attribute	Namespace of attribute	Name of attribute	Value of attribute	Example attribute-value pair
One	xmlns	xs	http://www.w3.org/2001/XMLSchema	xmlns:xs="http://www.w3.org/2001/XMLSchema"
One	xmlns	xsi	http://www.w3.org/2001/XMLSchema-instance	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
One	N/A	Version	<schema version number>	version="1..2.3"

NOTE 2 *rootPackageName* in Table 4 refers to the name of the corresponding <<xmlNamespace>> package of the root <<xmlSchema>> package.

8 Additional encodings

In order to maximise use of existing approaches and technologies, some classes are specifically encoded using relevant implementations from ISO or other organizations. These encodings are available at http://standards.iso.org/iso/19139/resources.

9 Encoding for modularity and reuse

9.1 UML packages and XML namespaces

In the process of encoding UML models into XML schema decisions must be made about relationships between UML packages and XML namespaces. Rules controlling the relationship between packages and namespaces are:

1) the XML implementation will include a minimum of one namespace per UML package in the conceptual model, i.e. multiple UML packages should not be combined into single namespaces;

2) UML packages may be split into multiple namespaces if necessary to ease the implementation and management of the life-cycles of various components;

3) exceptions to Rule 1 may be required to minimize dependencies between namespaces and eliminate circular dependencies.

9.2 UML model for XML implementation

Relationships between classes in different UML packages result in dependencies between packages that make reuse impossible without including both packages. In order to manage these relationships and facilitate modularization and automated schema generation, an XML-specific implementation layer may be added to the UML model without affecting the semantics. This layer includes

1) abstract classes that allow decoupling of model packages,

2) addition of tagged values and stereotypes required by the UML to XML transformation software,

3) refactoring of some model packages where required to eliminate circular dependencies between XML namespaces, and

4) abstract classes created to define substitution groups for classes that are, or might be, used by XML implementations of other ISO models.

Abstract classes for linkage between namespaces are all included in a single package.

Requirement /req/modularity/