```
〈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〉
```

例 2：图 14 所示的 AbsClass1 类的 XCGE 为：

```
〈xs:element name = "AbstractAbsClass1" type = "ns1:AbstractAbsClass1_Type" abstract = "true"/〉
```

例 3：图 14 所示的 AbsClass1 类的 XCPT 为：

```
〈xs:complexType name = "Class1_PropertyType"〉
    〈xs:sequence minOccurs = "0"〉
     〈xs:element ref = "ns1:AbstractAbsClass1"/〉
    〈/xs:sequence〉
    〈xs:attributeGroup ref = "gco:ObjectReference"/〉
    〈xs:attribute ref = "gco:nilReason"/〉
〈/xs:complexType〉
```

例 4：使用 GB/T 19710—2005 实际例子的 XCPT 样例如图 15 所示。

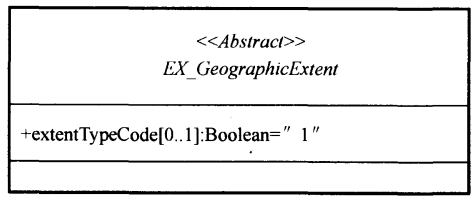

图 15　引自 GB/T 19710—2005 的 EX_GeographicExtent 抽象类的 UML

图 15 所示的 EX_GeographicExtent 抽象类的 XCT 为：

```
〈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〉
```

图 15 所示的 EX_GeographicExtent 抽象类的 XCGE 为：

```
〈xs:element name = "AbstractEX_GeographicExtent"
type = "gmd:AbstractEX_GeographicExtent_Type" abstract = "true"/〉
```

图 15 所示的 EX_GeographicExtent 抽象类的 XCPT 为：

```
〈xs:complexType name = "EX_GeographicExtent_PropertyType"〉
  〈xs:sequence minOccurs = "0"〉
    〈xs:element ref = "gmd: AbstractEX_GeographicExtent"/〉
  〈/xs:sequence〉
  〈xs:attributeGroup ref = "gco:ObjectReference"/〉
  〈xs:attribute ref = "gco:nilReason"/〉
〈/xs:complexType〉
```

8.5.3　继承关系与子类的编码

继承概念通过泛化这一简单扩展方法来实现。诚然这是对继承和子类概念的简化，因为忽略了包含约束的泛化关系和多重继承的泛化关系。这一仅用于扩展的泛化是有意设计成这样的，对于约束性泛化关系的实现在 A.4 讨论。就 XML 模式而言，不能支持多重继承，只能模拟实现。由于

GB/T 19710—2005没有多重继承的情况，多种继承的模拟机制这里不做考虑。

同默认的 UML 级相比，gmd 中所有具有相应 XCT，XCGE 和 XCPT 的子类与缺省类一样。子类的 XCT 与缺省类的区别包括：

a) XCT 的 xs:complexContent 元素内有一个 xs:extension 元素，其 base 属性等于基类或超类的受限命名空间 XCT(这不同于缺省 UML 类 xs:extension 元素的 XCT，其 base 属性是一个提供 id 和 uuid 属性的 gco:AbstractObject_Type 。通过扩展基类，所有必要的属性仍然存在)。注意如果超类为抽象类，则遵循 8.5.2 的要求，相应的 XCT 的名称前应加上“Abstract”。

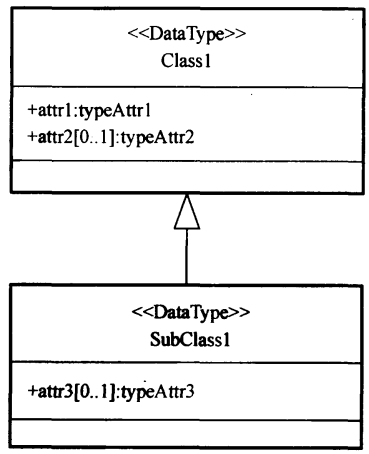

图 16 子类的 UML 样例

例 1：编码图 16 所示的类 SubClass1 的第 1 步为：

```
〈xs:complexType name = "SubClass1_Type"〉
  〈xs:complexContent〉
    〈xs:extension base = "ns1:Class1_Type"〉
      (…)
    〈/xs:extension〉
  〈/xs:complexContent〉
〈/xs:complexType〉
```

b) xs:extension 元素包含一个 xs:sequence 元素，该元素又包含 8.2 第 4 步中的缺省 UML 类列出的所有特性。注意该序列仅包含该子类专有的特性。

例 2：编码图 16 所示的类 SubClass1 的第 1 步为：

```
〈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〉
```

子类的 XCGE 和缺省 UML 类的 XCGE 的区别在于包含额外的 substitutionGroup 属性，其值等于基类的受限命名空间的 XCGE。

例 3：图 16 所示 SubClass1 类的 XCGE 编码为：

```
〈xs:element name = "SubClass1" type = "ns1:SubClass1_Type" substitutionGroup = "ns1:Class1"/〉
```

子类的 XCPT 与缺省 UML 类的 XCPT 编码方法相同。

例 4：图 16 所示 SubClass1 类的 XCPT 编码为：

```
〈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〉
```

例 5：使用 GB/T 19710—2005 实际例子的 XML 模式内的子类样例如图 17 所示。

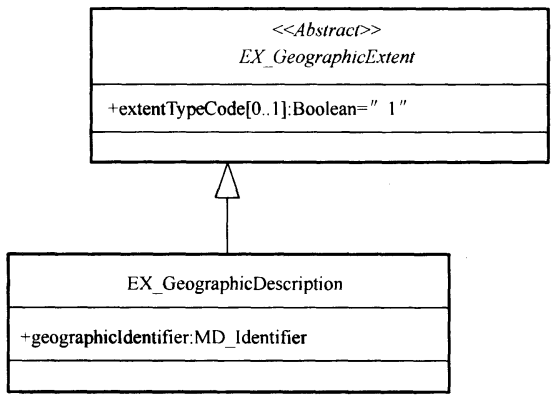

图 17　引自 GB/T 19710—2005 的 EX_GeographicExtent 和 EX_GeographicDescription 类的 UML

图 17 中 EX_GeographicDescription 类的 XCT 为：

```
〈xs:complexType name = "EX_GeographicDescription_Type"〉
    〈xs:complexContent〉
        〈xs:extension base = "gmd:AbstractEX_GeographicExtent_Type"〉
        〈xs:sequence〉
         〈xs:element name = "geographicIdentifier" type = "gmd:MD_Identifier_PropertyType"/〉
        〈/xs:sequence〉
        〈/xs:extension〉
    〈/xs:complexContent〉
〈/xs:complexType〉
```

图 17 中 EX_GeographicDescription 类的 XCGE 为：

```
〈xs:element name = "EX_GeographicDescription"
type = "gmd:EX_GeographicDescription_Type"
substitutionGroup = "gmd:AbstractEX_GeographicExtent"/〉
```

图 17 中 EX_GeographicDescription 类的 XCPT 为：

```
〈xs:complexType name = "EX_GeographicDescription_PropertyType"〉
     〈xs:sequence minOccurs = "0"〉
〈xs:element ref = "gmd:EX_GeographicDescription"/〉
    〈/xs:sequence〉
    〈xs:attributeGroup ref = "gco:ObjectReference"/〉
    〈xs:attribute ref = "gco:nilReason"/〉
〈/xs:complexType〉
```

8.5.4　枚举的编码

地理信息国家标准的概念模式语种支持两种列举类型，在其定义中都给出了一个符合要求的助记符清单[ISO/TS 19103]。这两种类型是枚举型(Enumeration)和代码表(CodeList)，注意这两种类型是不同的。构造型为 Enumeration 的类只能包含表示枚举值的简单属性[ISO/TS 19103]，〈〈Enumeration〉〉类只用于不需要对值列表进行扩充的情况，否则需要使用〈〈CodeList〉〉类。

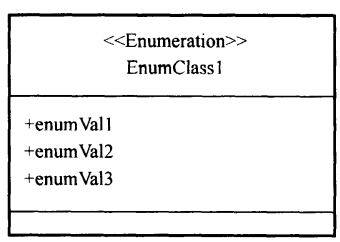

图 18　构造型为枚举类型的类的 UML 样例

a) Enumeration 类的代码开始于创建一个 XCT，以 xs:simpleType 元素开始并有一个"name"属性等同于 UML 类名加_Type 后缀。

例 1：步骤 a) 创建图 18 的 EnumClass1 类：

```
〈xs:simpleType name = "EnumClass1_Type"〉
 (…)
〈/xs:simpleType〉
```

b) xs:simpleType 元素包含一个 xs:restriction 元素并有一个"base"属性，其值为"xs:string".

例 2：步骤 b)创建如图 18 的 EnumClass1 类：

```
〈xs:simpleType name = "EnumClass1_Type"〉
〈xs:restriction base = "xs:string"〉
 (…)
〈/xs:restriction〉
〈/xs:simpleType〉
```

c) 每个 xs:restriction 元素包含一系列 xs:enumeration 元素，其 value 属性等于枚举 UML 图中的属性值。

例 3：步骤 c)创建如图 18 的 EnumClass1 类：

```
〈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〉
```

〈〈Enumeration〉〉类的 XCGE 遵循默认的 XCGE 规则并有 substitutionGroup 属性等于 gco:CharacterString。

例 4：图 18 中的 EnumClass1 类的 XCGE 为：

```
〈xs:element name = "EnumClass1" type = "ns1:EnumClass1_Type"
 substitutionGroup = "gco:CharacterString"/〉
```

〈〈Enumeration〉〉类的 XCPT 遵循 8.4.2 中规定的简单类型默认 XCT 编码方法。

例 5：图 18 中 EnumClass1 类的 XCPT 为：

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

例 6：GB/T 19710—2005 中的一个具体枚举类型编码示例如图 19 所示。

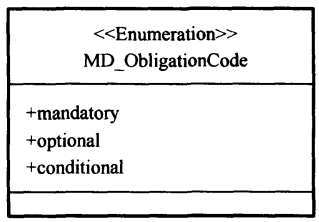

图 19 引自 GB/T 19710—2005 的 MD_ObligationCode 类

图 19 中 MD_ObligationCode 类的 XCT 为：

```
〈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>
```

图 19 中 MD_ObligationCode 类的 XCGE 为：

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

图 19 中 MD_ObligationCode 类的 XCPT 为：

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

8.5.5 代码表的编码

8.5.5.1 代码表编码的构成部分

构造型为代码表的类也是列举型的类型，与构造型为枚举型的类相似。其区别在于〈〈CodeList〉〉类可以扩充。GB/T 19710—2005 附录 B 中定义的所有〈〈CodeList〉〉类都在包含名称、代码取值和定义三列的表格中进行了描述。

表 1 引自 GB/T 19710—2005 的描述 CI_DateTypeCode〈〈CodeList〉〉的表格

	名称(中文)	名称(英文)	代码取值	定义
1	CI_DateTypeCode	CI_DateTypeCode	DateTypCd	标识给定事件发生的时间
2	生产	creation	001	标识资源完成生产的日期
3	出版	publication	002	标识资源出版的日期
4	修订	revision	003	标识资源检查、重新检查、改善或更新的日期。

描述代码表的表格每一行描述代码表中一个唯一的概念或选项。构造型 CodeList 的专门编码是用来支持 GB/T 19710—2005 制定者的期望的特性，包括：

a) 代码表及其相关定义通过注册表进行控制。GB/T 19710—2005 附录 B 是 GB/T 19710—2005 代码表的基本概念性注册表，也是建立用户自己的注册表的基本来源。

b) GB/T 19710—2005 附录 B 中描述代码表的表格的名称列，包含了可被所有软件使用的值，用于识别代码表中定义的概念或选项。因此，其值不能特定于任何语种，即使是英语也不行。

c) 用户需要使用这里指定为代码空间的一种或多种自然语言、方言或成语等来表达代码表中的各个唯一性概念或选项。每种代码空间中，都可给出名称或定义，如果代码空间是自然语言，该名称或定义是以给定语区(语言、国家以及字符集)的规则进行表达(这类似于在 GB/T 19710—2005 附录 B 的代码表表格中增加列)。

d) 用户可能需要添加 GB/T 19710—2005 没有考虑到的新概念或选项，但需要在用户注册表中对这种扩充进行必要控制(这类似于在 GB/T 19710—2005 附录 B 的代码表表格中增加行)。

为了满足对代码表的这些特性的期望，建立一个专门的“CodeListValue_Type” XCT，包含三个属性：

```
<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:attribute name = "codeSpace" type = "xs:anyURI"/>
 </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

该专门 XCT 的定义使得任何代码表都可以替换字符串类型。针对 8.5.5.2 中描述的每一个代码

表都定义了一个基于该专门 XCT 的 XCGE 和一个 XCPT。因此，任何代码表实例都是一个 XML 元素，其名称为 XCGE 的名称，该元素包含一个值和该专门 XCT 定义的三个 XML 属性。

codeList 属性包含一个指向某一注册表内的代码表定义或代码表目录的 URL。

例 1：如果表 1 的内容包含在 ISO/TC 211 网站上的一个代码表目录中，则 codeList 属性的值为：

codeList = "http://www.tc211.org/ISO 19139/resources/codeList.xml#CI_DateTypeCode"

属性 codeListValue 包含代码表值的标识符，即 GB/T 19710—2005 附录 B 的表格中的名称列中给出的值。代码表目录或注册表需要包含该值的一个明确的名称和定义，既使用元数据的默认语言，也使用其他代码空间，某些代码空间可能对应元数据支持的不同语区。

例 2：如果某个实例文档的一段元数据要说明生产完成时间，则基于表 1，codeListValue 的值为：

codeListValue = "creation"

每一个替代表达方式都应通过 codeSpace 给出其代码空间。codeSpace 属性是一个可选的标识符(URI)，指向表示为元素值的代码表值的替代表达方式。域代码的 codeSpace URI 值为“domainCode”。

例 3：如果某个实例文档的一段元数据要说明生产完成时间，但希望使用域代码表示，则基于表 1，codeSpace 的值为：

codeSpace = "domainCode"

如果 codeSpace 属性不出现，元素的内容要么是给定代码空间代码表值的名称，要么是元数据默认语言下代码表值的名称。

例 4：下面是一个使用 domaineCode 代码空间表达的 CI_DateTypeCode 的完整实例：

```
〈CI_DateTypeCode
codeList = "http://www.tc211.org/ISO 19139/resources/codeList.xml#CI_DateTypeCode" codeListValue = "creation" codeSpace = "domainCode"〉001〈/CI_DateTypeCode〉
```

例 5：下面是使用元数据默认语言（英语）表示的 CI_DateTypeCode 的完整实例：

```
〈CI_DateTypeCode
codeList = http://www.tc211.org/ISO 19139/resources/codeList.xml#CI_DateTypeCode
codeListValue = "creation"〉Creation〈/CI_DateTypeCode〉
```

XML 元素值保证了用户可以访问实际代码表值的有效默认表达结果，三个属性保证系统可以获得代码表的定义及其值，特别是用于定制用户界面的时候（如生成一个下拉框以提供代码表值选择列表，多语种或多文化管理等）。

8.5.5.2 代码表编码细节

在 8.5.5.1 中介绍了 CodeListValue_Type，该复杂类型作为所有〈〈CodeList〉〉类的 XCT。〈〈CodeList〉〉类的 XCGE 编码为一个全局元素，其 name 属性等于 UML 类的名称，type 属性为 CodeListValue_Type。与〈〈Enumeration〉〉类相似，该 XCGE 也包含一个 substitutionGroup 属性，其值等于"gco:CharacterString"。

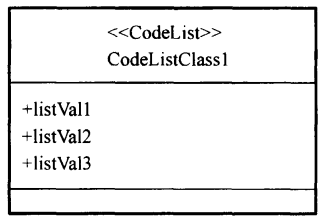

图 20 构造型为 CodeList 的类的 UML 样例

例 1：图 20 中 CodeListClass1 类的 XCGE：

```
〈xs:element name = "CodeListClass1" type = "gco:CodeListValue_Type"
substitutionGroup = "gco:CharacterString"/〉
```

〈〈CodeList〉〉类的 XCPT 遵循 8.4.2 中针对简单类型定义的编码规则。

例 2：图 20 中 CodeListClass1 类的 XCPT：

```
〈xs:complexType name = "CodeListClass1_PropertyType"〉
    〈xs:sequence minOccurs = "0"〉
```

```
      〈xs:element ref = "gmd:CodeListClass1"/〉
    〈/xs:sequence〉
    〈xs:attribute ref = "gco:nilReason"/〉
〈/xs:complexType〉
```

例 3：GB/T 19710—2005 中实际代码表类的编码(见图 21)：

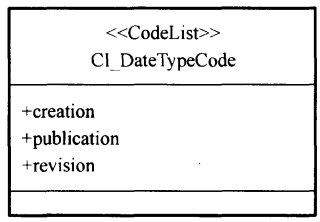

图 21　引自 GB/T 19710—2005 的 CI_DateTypeCode 类

图 21 中 CI_DateTypeCode 类的 XCGE 为：

```
〈xs:element name = "CI_DateTypeCode" type = "gco:CodeListValue_Type"
substitutionGroup = "gco:CharacterString"/〉
```

图 21 中 CI_DateTypeCode 类的 XCPT 为：

```
〈xs:complexType name = "CI_DateTypeCode_PropertyType"〉
   〈xs:sequence minOccurs = "0"〉
     〈xs:element ref = "gmd:CI_DateTypeCode"/〉
   〈/xs:sequence〉
   〈xs:attribute ref = "gco:nilReason"/〉
〈/xs:complexType〉
```

8.5.6　联合的编码

所有构造型为 Union 的类都通过一个 xs:complexType 转换为 XCT，其 name 属性等于 UML 中的类名加上"_Type"。在 xs:complexType 元素中有一个 xs:choice 元素，对应于 UML 类中的每一个属性，该 xs:choice 元素包含一个相应的 xs:element 元素。每一个 xs:element 元素的 name 属性等于 UML 类中的属性名称，type 属性等于属性的类型前面加上合适的命名空间前缀，后面加上"_PropertyType"。

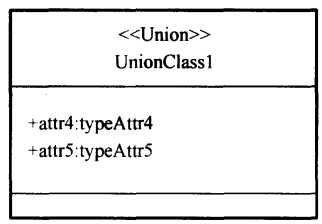

图 22　构造型为 Union 的类的 UML 样例

例 1：图 22 中的 UnionClass1 类的 XCT：

```
<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>
```

〈〈Union〉〉类的 XCGE 编码遵循 8.3 描述的缺省编码规则。由于联合(Union)也属不宜采用引用的情况，其 XCPT 遵循 8.4.2 中描述的简单类型的编码规则。

例 2：图 22 中的 UnionClass1 类的 XCGE：

```
〈xs:element name = "UnionClass1" type = "ns1:UnionClass1_Type"/〉
```

例 3：图 22 中的 UnionClass1 类的 XCPT：

```
    <xs:complexType name = "UnionClass1_PropertyType">
      <xs:sequence minOccurs = "0">
        <xs:element ref = "gmd:UnionClass1"/>
      </xs:sequence>
```

```
    <xs:attribute ref = "gco:nilReason"/>
  </xs:complexType>
```

例 4：基于 GB/T 19710-2005 实际例子的联合类型类样例如图 23 所示。

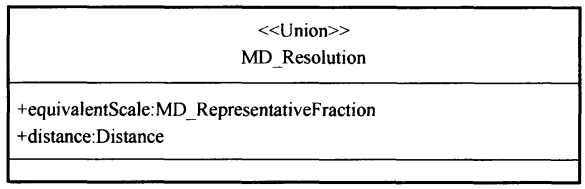

图 23　引自 GB/T 19710—2005 的 MD_Resolution 类

图 23 中的 MD_Resolution 类的 XCT：

```
<xs:complexType name = "MD_Resolution_Type">
  <xs:choice>
    <xs:element name = "equivalentScale"
    type = "gmd:MD_RepresentativeFraction_PropertyType"/>
    <xs:element name = "distance" type = "gco:Distance_PropertyType"/>
  </xs:choice>
</xs:complexType
```

图 23 中的 MD_Resolution 类的 XCGE：

```
〈xs:element name = "MD_Resolution" type = "gmd:MD_Resolution_Type"/〉
```

图 23 中的 MD_Resolution 类的 XCPT：

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

8.5.7　元类的编码

虽然 GB/T 19710—2005 中没有构造型为 MetaClass(元类)的类，但包含一些与构造型是元类的类相关的特性。对于这种特性，相对应的类只能根据一种实现进行实例化。编码方法和对实现方法的使用在 8.5.8 中进一步讨论。

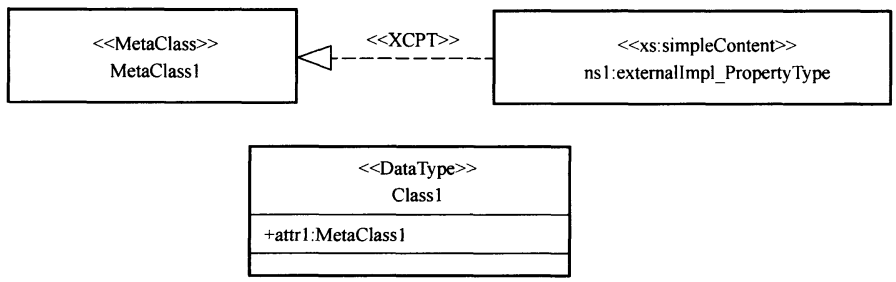

图 24　构造型为元类的类的 UML 样例

确定了缺省编码 XCT 的序列中某一元素的 type 属性后，再使用〈〈MetaClass〉〉类的实现。换言之，图 24 中 Class1 的 XCT，“attr1”元素的类型是“ns1：externalImpl_PropertyType”，而不是“MetaClass1_PropertyType”，因为图 24 表明 ns1：externalImpl_PropertyType 与 MetaClass1 之间是实现关系。

例：图 24 中 Class1 类的 XCT 为：

```
<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>
```

8.5.8 外部并入实现模式的编码

8.5.8.1 外部并入实现模式编码介绍

6.5 中介绍了使用外部并入实现模式可以改善本指导性技术文件的互操作性和可用性。充分利用地理信息国家标准已有实现或针对某项特殊编码技术使用外部实现是可能的。UML 中实现关系的 UML 符号用来表示何处使用其他地理信息国家标准中定义的 XML 模式。

有三种途径把外部实现纳入到地理信息国家标准描述的 UML 模型概念中。地理信息国家标准的 UML 类的名称是通向实现模式的入口，实现模式基于 8.5.8.2 和 8.5.8.3 描述的实现(realization)的构造型。

8.5.8.2 通过 XCPT 编码

使用外部编码最简单的情况是已有的实现已经提供了对应地理信息国家标准中 UML 类的类类型、全局元素和类特性类型，并符合 8.2、8.5 描述的缺省编码规则。在 UML 中，这通过在如图 25 所示的实现关系上提供一个 XCPT 构造型来表示。

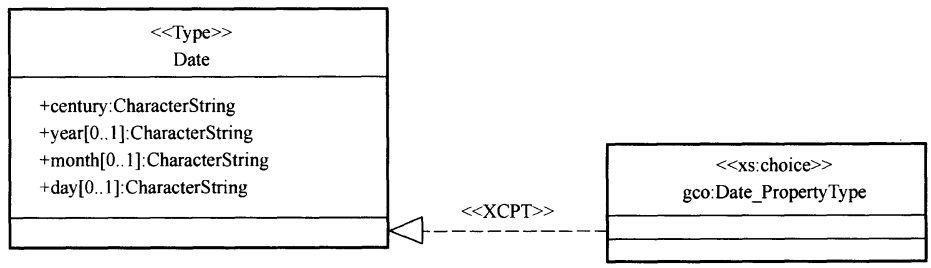

图 25 ISO/TS 19103 的日期类型通过 gco:Date_PropertyType 实现

这种情况下，不需为目标类生成 XCT、XCGE 和 XCPT。如果目标类是某一 XCT 的特性类型，则使用源类的名称作为 type 属性的值。大多数情况下，type 属性的值将遵循缺省编码规则，即使用目标类的名称加"_PropertyType"，但并非总是这样。

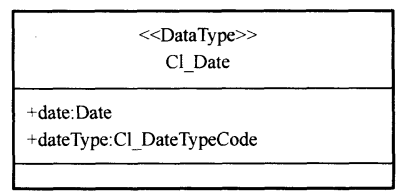

图 26 GB/T 19710—2005 中的 CI_Date 类

例：图 26 中 CI_Date 类的 XCT 为：

```
<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 = "gmd:CI_DateTypeCode_PropertyType"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

8.5.8.3 通过 XCGE 编码

另一种需要使用外部编码的情况是已有实现提供了可用的类类型和全局元素，但需要保留外部实现的继承层次关系。这种情况下，需要使用地理信息国家标准 UML 模型的类名称生成一个 XCPT。该 XCPT 通过外部实现的 XCGE 用作该实现模式的入口。在 UML 中，这通过在如图 27 所示的实现

关系上提供一个 XCGE 构造型来表示。

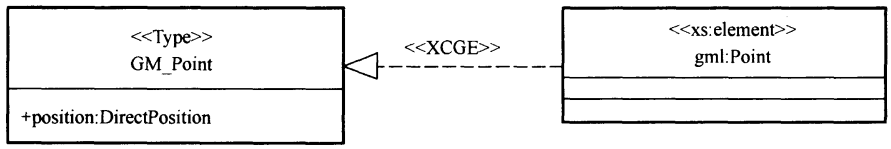

图 27　GB/T 23707—2009 的 GM_Point 通过 gml:Point 实现

如果实现关系有一个 XCGE 构造型，目标类的 XCPT 遵循 8.4 针对 XCPT 编码描述的编码规则，并且使用实现关系的源类的名称作为 xs:element 中 ref 属性的值。

例 1：图 27 中 GM_Point 的 XCPT：

```
<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>
```

某些情况下，表示外部实现的 UML 所包含特性的类型可能是构造型为 xs:element 的类。这样的话，需要一个对应于该类的 XCGE，在 UML 图中，其位于一个类名命名空间前缀表明的命名空间中。该 XCGE 将使用 xs:element 来定义，该元素的 name 属性等于 UML 中所示 XML 类的名称减去命名空间前缀，type 属性等于 UML 中仅有的命名特性。xs:element 的任何其他属性表示为 XML 类的特性，在 UML 中，其作为特性的类型，并有一个缺省值。

例 2：gco 命名空间中 xs:element 的 XCGE 样例如图 28 所示。

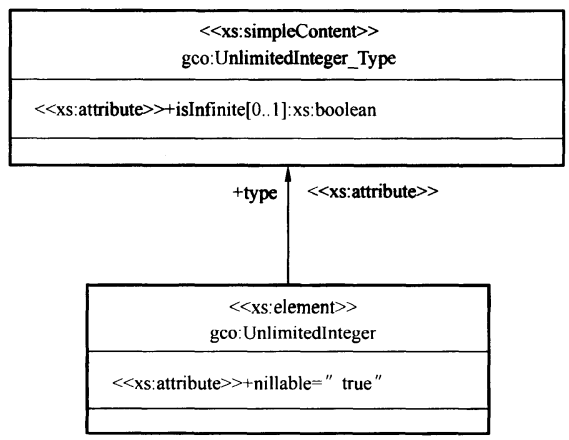

图 28　构造类为 xs:element 的 gco:UnlimitedInteger 的 XCGE

图 28 中无限整数的 XCGE 为：

```
〈xs:element name = "UnlimitedInteger" type = "gco:UnlimitedInteger_Type" nillable = "true"/〉
```

8.5.8.4　通过 XCT 编码

8.5.8.4.1　概述

通常在外部实现中只提供一个 XCT，这种情况下使用 simpleType。此时，地理信息国家标准的类到 XML 模式对象的实现关系带一个 XCT 构造型。

8.5.8.4.2　构造型 xs:simpleType

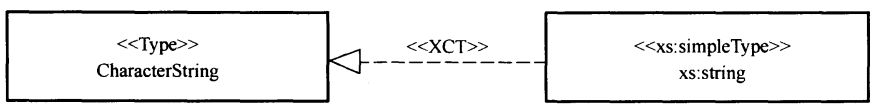

图 29　使用 xs:string 实现 CharacterString

如果实现关系带一个 XCT 构造型，同时源类具有一个 xs:simpleType 构造型，则 XCGE 遵循 8.3

针对 XCGE 编码描述的编码规则，并使用对应 XCT 的名称作为 type 属性的值。

例：图 29 中 CharacterString 的 XCGE：

```
〈xs:element name = "CharacterString" type = "xs:string"/〉
```

8.5.8.4.3 构造型 xs:simpleContent

如果实现关系带一个 XCT 构造型，同时源类具有一个 xs: simpleContent 构造型，针对该元类生成一个 XCT。该 XCT 遵循以下规则：

a) 生成一个 xs:complexType 元素，其 name 属性等于去掉命名空间前缀的 XML 类的名称。如果碰巧该 XML 类是抽象类，则 xs:complexType 将包含一个 abstract 属性，其值为“true”。

b) xs:complexType 元素包含一个 xs:simpleContent 元素。

c) xs:simpleContent 包含一个 xs:extension 元素，其 base 属性值设为外部实现超类的名称。

d) 根据表示 XML 类的 UML 所示的特性及各自的构造型，xs:simpleContent 元素会包含 xs:attribute 或 xs:attributeGroup 元素。如果 UML 特性中有一个类型值，则 xs:attribute 或 xs:attributeGroup 元素会有一个 name 属性，其值等于 UML 中特性的名称，还有一个 type 属性，其值为 UML 中所示的类型。如果 UML 中的特性没有类型，则 xs:attribute 或 xs:attributeGroup 会包含一个 ref 属性，其值等于 UML 中特性的名称。与 8.2 中的缺省 XCT 编码一样，minOccurs 和 maxOccurs 的值与 ISO 19118:2005 表 A.5 一致。

例：gco 命名空间中一个 xs:simpleContent 的 XCT 实现如图 30 所示。

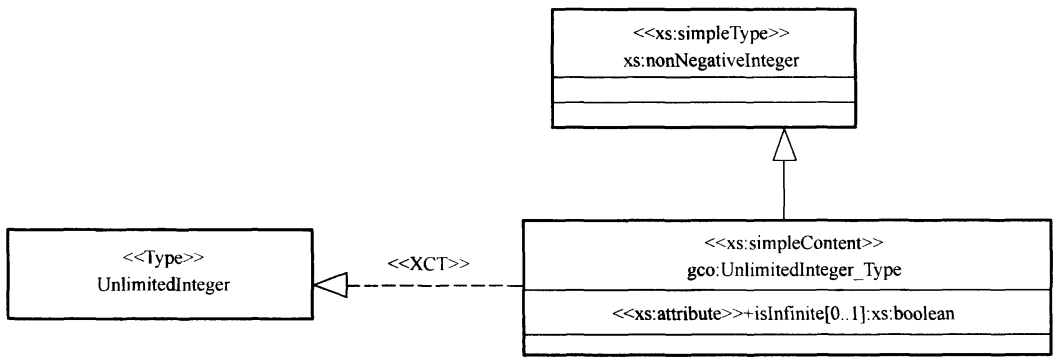

图 30 构造型为 xs:simpleContent 的 UnlimitedInteger_Type 的 XCT 实现

图 30 中 UnlimitedInteger 的 XCT：

```
<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>
```

8.5.8.4.4 构造型 xs:complexType

如果实现关系中有一个 XCT 构造型，并且源类有一个 xs:complexType 构造型，则根据源类生成一个 XCT。该 XCT 需符合以下要求：

a) 生成一个 xs:complexType，其 name 属性等于去掉命名空间前缀的 XML 类名称。如果 XML 类是抽象类，则 xs:complexType 应包含一个 abstract 属性，其值设为“true”。

b) xs:complexType 包含一个 xs:complexContent 元素。

c) xs:complexContent 元素包含一个 xs:extension 元素，其 base 属性设为外部实现超类的名称。如果没有外部实现超类，则 base 属性设为 gco:AbstractObject。

d) xs:complexContent 元素包含一个 xs:sequence 元素，它由若干对应于 XML 特性的 xs:element 组成，这些特性的构造型为 xs:element。每一个 xs:element 元素的 name 属性等于特性

的名称，type 属性等于属性的类型。与 8.2 中的缺省 XCT 编码一样，minOccurs 和 maxOccurs 的值与 ISO 19118:2005 表 A.5 一致。

e) 在 xs:sequence 之后，视 XML 类中构造型为 xs:attribute 或 xs:attributeGroup 特性的情况，可以有 xs:attribute 或 xs:attributeGroup 元素。如果 UML 元素中有类型值，则 xs:attribute 或 xs:attributeGroup 元素的 name 属性的值等于 UML 中元素的名称，type 属性等于 UML 中显示的类型。如果没有类型值，则 xs:attribute 或 xs:attributeGroup 元素包含一个 ref 属性，其值等于 UML 中元素的名称。与 8.2 中的缺省 XCT 编码一样，minOccurs 和 maxOccurs 的值与 ISO 19118:2005 表 A.5 一致。

例：gmd 命名空间中的一个 xs:complexType 的 XCT 实现实例如图 31 所示。

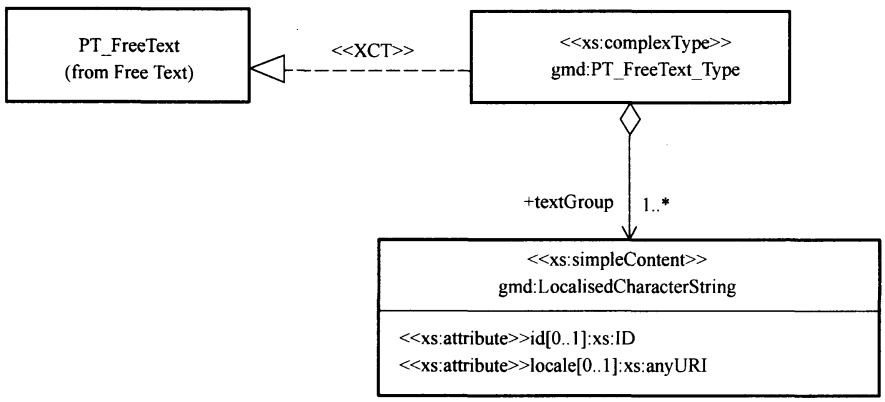

图 31 构造型为 xs:complexType 的 gmd:PT_FreeText_Type 的 XCT 实现

图 31 中 PT_FreeText 的 XCT：

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

8.5.8.4.5 构造型 xs: union

如果一个类有 xs: union 构造型，根据 UML 类生成一个 XCT，该 XCT 为一个 xs:simpleType，其 name 属性等于 UML 类的名称。在 xs:simpleType 中是一个 xs:union，其 member 属性等于 UML 类中所有元素的名称，这些名称用空格隔开。

例：gco 命名空间中 xs:union 的 XCT 实例如图 32 所示。

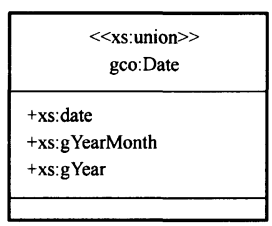

图 32 构造型为 xs:union 的 gco:Date 的 XCT

图 32 中 Date 的 XCT：

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

8.5.8.4.6 通过 XCT 编码时生成 XCGE

对任何外部实现 XCT 生成一个 XCGE，这些 XCGE 遵循 8.3、8.5.2(针对抽象类)和 8.5.3(针对子类)中的缺省编码规则。

8.5.8.4.7 生成 XCPT

如果实现关系的源类的构造型为 xs:complexType，则根据 8.4.1 中的编码规则生成 XCPT，如果其构造型为 xs:simpleType 或 xs:simpleContent.，则根据 8.4.1 中的编码规则生成 XCPT。

例 1：图 29 中，CharacterString 的 XCPT 为：

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

如果类的构造型为 xs:choice，则对生成 XCPT 的缺省编码规则稍作修改。假设构造型为 xs:choice 的类代表一个唯一的 XCPT，这样其名称则应总是包含后缀"_PropertyType"。该名称将被用作 XCPT 的 xs:complexType 元素 name 属性的值，该 XCPT 也包含一个 minOccurs 属性等于 0 的 xs:choice 元素。xs:choice 中包含若干对应于每个 UML 类特性的 xs:element 元素。每个 xs:element 都包含一个 ref 属性，等于指定为 UML 类特性的 XCGE 的名称。xs:choice 结束标记之后，有一个 gco:nilReason 然后 XCPT 就结束了。

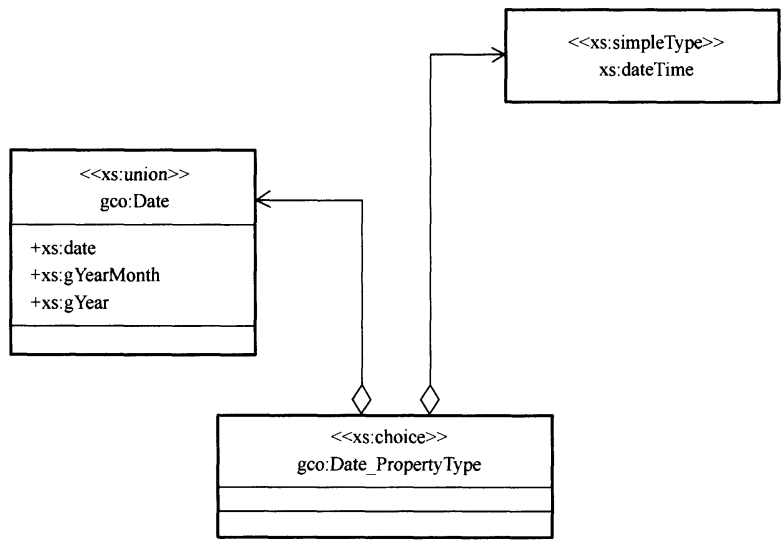

图 33 构造型为 xs:choice 的类的 UML 样例

例 2：图 33 中 gco:Date_PropertyType 的 XCPT：

```
<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>
```

8.6 XML 命名空间包的编码

在 5.4 介绍了构造型 xmlNamespace，它是组织到同一命名空间中的一组 XML 对象。UML 包可以包含两个模型元素之间的依赖关系，对其中的一个模型元素的变动会影响另外一个[ISO/TS 19103]。对于〈〈xmlNamespace〉〉包，需要考虑两种类型的依赖关系。依赖关系上的构造型 implement 具有一个代表 XML 命名空间(或命名空间前缀)的源，该命名空间实现目标包中的抽象概念。6.1给出了 gmd 的介绍，图 34 使用 UML 的〈〈xmlNamespace〉〉包和〈〈implement〉〉依赖关系说明了6.1

文字介绍的内容。

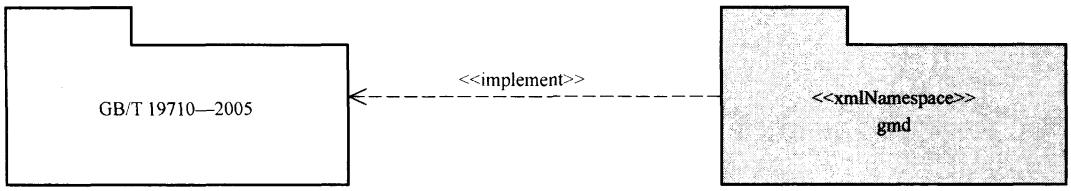

图 34　xml 命名空间与实现

依赖关系上的构造型 import 有一个代表分组到同一命名空间内的对象的源，这些对象又依赖于另一个命名空间内的对象。如果两个为〈〈xmlNamespace〉〉包〈〈implement〉〉依赖关系的目标的抽象包之间存在依赖关系，则〈〈import〉〉依赖关系存在于两个〈〈xmlNamespace〉〉包之间。图 35 中示出了一个 gmd 和 gss 包之间的〈〈import〉〉依赖关系。

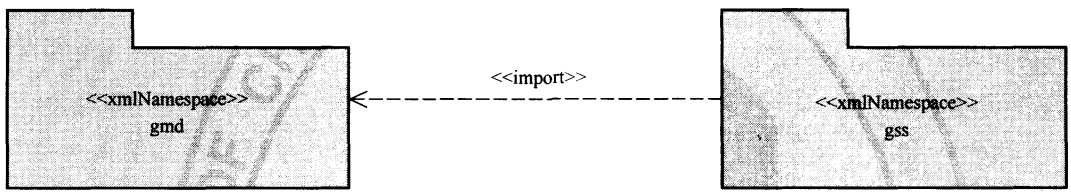

图 35　xmlNamespace 与导入

〈〈xmlNamespace〉〉包不是直接编码成 XML 的，但它有相应的〈〈xmlSchema〉〉包利用〈〈implement〉〉和〈〈import〉〉依赖关系来生成符合要求的 XML 模式。

8.7　XML 模式包的编码

5.4 中，构造型为 xmlSchema 的包是代表 XML 模式的包。第 9 章中 UML 内任何〈〈xmlSchema〉〉包都存在一个相应的 XML 模式文件，其名称与包的名称相同。〈〈xmlSchema〉〉包可严格按照 8.6 针对〈〈xmlSchema〉〉所说的那样使用〈〈implement〉〉依赖关系，也可使用〈〈include〉〉依赖关系。每个〈〈xmlSchema〉〉包都是以下两种类型之一。

第一种是根〈〈xmlSchema〉〉包，直接对应于〈〈xmlNamespace〉〉包，其名称为〈〈xmlNamespace〉〉包的名称加上".xsd"后缀。这意味着如图 35 所示有一个 gmd 〈〈xmlNamespace〉〉包，就有一个相应的构造型为 xmlSchema 的 gmd.xsd 包和一个作为本指导性技术文件一部分的 gmd.xsd 文件。

每一个 XML 模式文件都以〈?xml version="1.0" encoding="utf-8"?〉开头。同一文件中的根元素是 xs:schema，包含的属性如表 2 所示：

表 2　根〈〈xmlSchema〉〉包中 xs:schema 元素的属性

出现次数	名称空间	属性名	属性值	属性与值实例
1	空	targetNamespace	http://www.isotc211.org/2005/+包名	targetNamespace="http://www.isotc211.org/2005/gmd"
1	xmlns	包名	http://www.isotc211.org/2005/+包名	xmlns:gmd="http://www.isotc211.org/2005/gmd"
0 或多	xmlns	〈〈import〉〉依赖关系的目标包的包名	http://www.isotc211.org/2005/+〈〈import〉〉依赖关系的目标包的包名	xmlns:gco="http://www.isotc211.org/2005/gco"
1	xmlns	xs	http://www.w3.org/2001/XML Schema	xmlns:xs="http://www.w3.org/2001/XMlL Schema"
1	空	version	1.0	version="1.0"

注：packageName 指相应根〈〈xmlSchema〉〉包的〈〈xmlNamespace〉〉包的名称。

例:图 35 所示的 UML 对应一个 gmd. xsd 文件,其 xs:schema 申明如下所示:

```
〈 xs:schema targetNamespace = "http://www.isotc211.org/2005/gmd"
xmlns:gmd = "http://www.isotc211.org/2005/gmd" xmlns:gss = "http://www.isotc211.org/2005/gss"
xmlns:xs = "http://www.w3.org/2001/XMLSchema" version = "1.0" /〉
```

第二种〈〈xmlSchema〉〉包称作常规〈〈xmlSchema〉〉包,直接对应地理信息国家标准中已有的包,通过带 implement 构造型的依赖关系来表示。图 36 所示的样例表示一个常规〈〈xmlSchema〉〉包及其目标 GB/T 19710—2005 包。

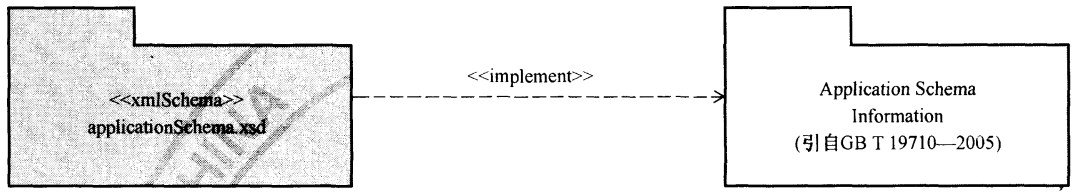

图 36 常规 xmlSchema 包

所有常规 xmlSchema 包正好有一个相应的根〈〈xmlSchema〉〉包,它或者是〈〈xmlSchema〉〉包之间的〈〈include〉〉依赖关系的源,或者是常规〈〈xmlSchema〉〉的根〈〈xmlSchema〉〉,它是到相关常规〈〈xmlSchema〉〉的〈〈include〉〉依赖关系的源。图 37 中,根〈〈xmlSchema〉〉为 root. xsd,常规〈〈xmlSchema〉〉为 schema1. xsd 和 schema2. xsd。root. xsd 〈〈xmlSchema〉〉是两个常规〈〈xmlSchema〉〉的根。

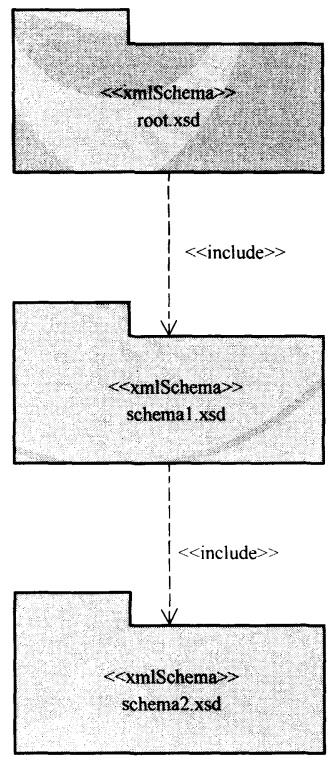

图 37 〈〈xmlSchema〉〉包之间的层级关系

每个 XML 模式文件中基于已有常规〈〈xmlSchema〉〉包生成的的第一个 XML 模式元素是〈?xml version="1.0" encoding="utf-8"?〉。同一文件中的第二个元素为 xs:schema,包含的属性如表 3 所示。

表 3　常规〈〈xmlSchema〉〉包中 xs:schema 元素的属性

出现次数	名称空间	属性名	属性值	属性与值实例
1	空	targetNam-espace	http://www.isotc211.org/2005/+包名	targetNamespace=" http://www.isotc211.org/2005/gmd"
1	xmlns	根包的名称	http://www.isotc211.org/2005/+根包的名称	xmlns:gmd=" http://www.isotc211.org/2005/gmd"
0 或多	xmlns	〈〈import〉〉依赖关系目标的根包包名	http://www.isotc211.org/2005/+〈〈import〉〉依赖关系目标的根包名称	xmlns:gco=" http://www.isotc211.org/2005/gco"
1	xmlns	xs	http://www.w3.org/2001/XML Schema	xmlns:xs="http://www.w3.org/2001/XMlL Schema"
1	xmlns	xsi	http://www.w3.org/2001/XML Schema-instance	xmlns:xs="http://www.w3.org/2001/XMlL Schema-instance"
1	空	version	1.0	version="1.0"
注：根包名称指相应根〈〈xmlSchema〉〉包的〈〈xmlNamespace〉〉包的名称。				

9　编码说明

9.1　编码说明概述

GB/T 19710—2005 及相关标准的实现遵循第 8 章叙述的编码规则。例外情况及基于外部类型的实现在本章详细说明。本章使用地理信息国家标准中通用的 UML 符号，加上 5.4 中定义的实现(realization)概念和实现(implementation)方面的若干构造型。

9.2　XML 命名空间

图 38 表示用于实现 GB/T 19710—2005 的不同命名空间(灰框部分)，包括各个命名空间之间的关系，还有地理信息国家标准包(白框部分)。

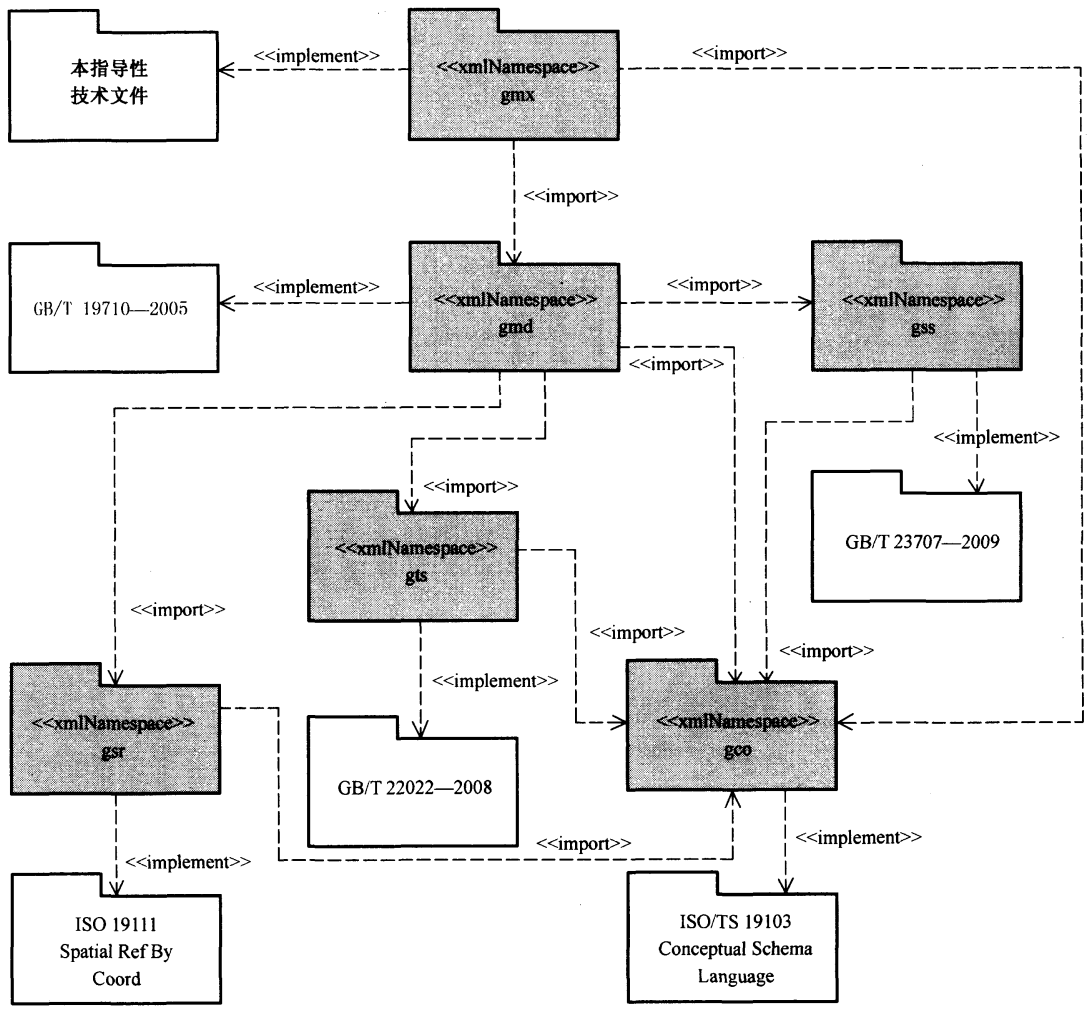

图 38　XML 包及相互关系

9.3　gmd 命名空间

9.3.1　gmd 命名空间的构成

该命名空间包含对 GB/T 19710—2005 的实现。命名空间的根是 gmd. xsd。图 39 详示了 gmd 命名空间的组成情况。

9.3.2　gmd. xsd

该 XML 模式直接或间接包含实现的 gmd 命名空间中所有概念，但不包含任何类型的申明。

9.3.3　metadataApplication. xsd

该 XML 模式实现了 GB/T 19710—2005 的 6.2 中定义的 UML 概念模式，包括以下类的实现：DS_Aggregate，DS_Dataset，DS_OtherAggregate，DS_Series，DS_Initiative，DS_Platform，DS_Sensor，DS_ProductionSeries，DS_StereoMate。

该 XML 模式中实现的类遵循第 8 章的编码规则。

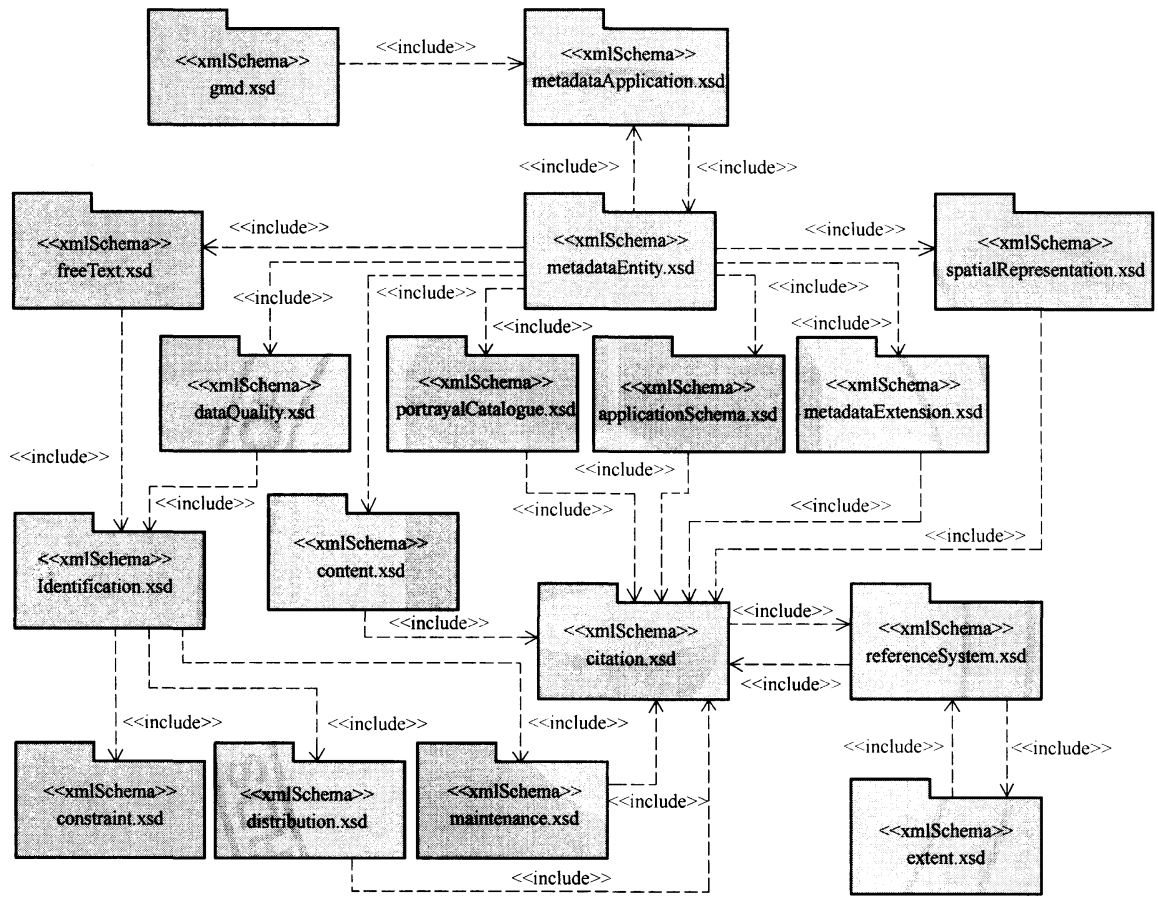

图 39 gmd 命名空间的构成

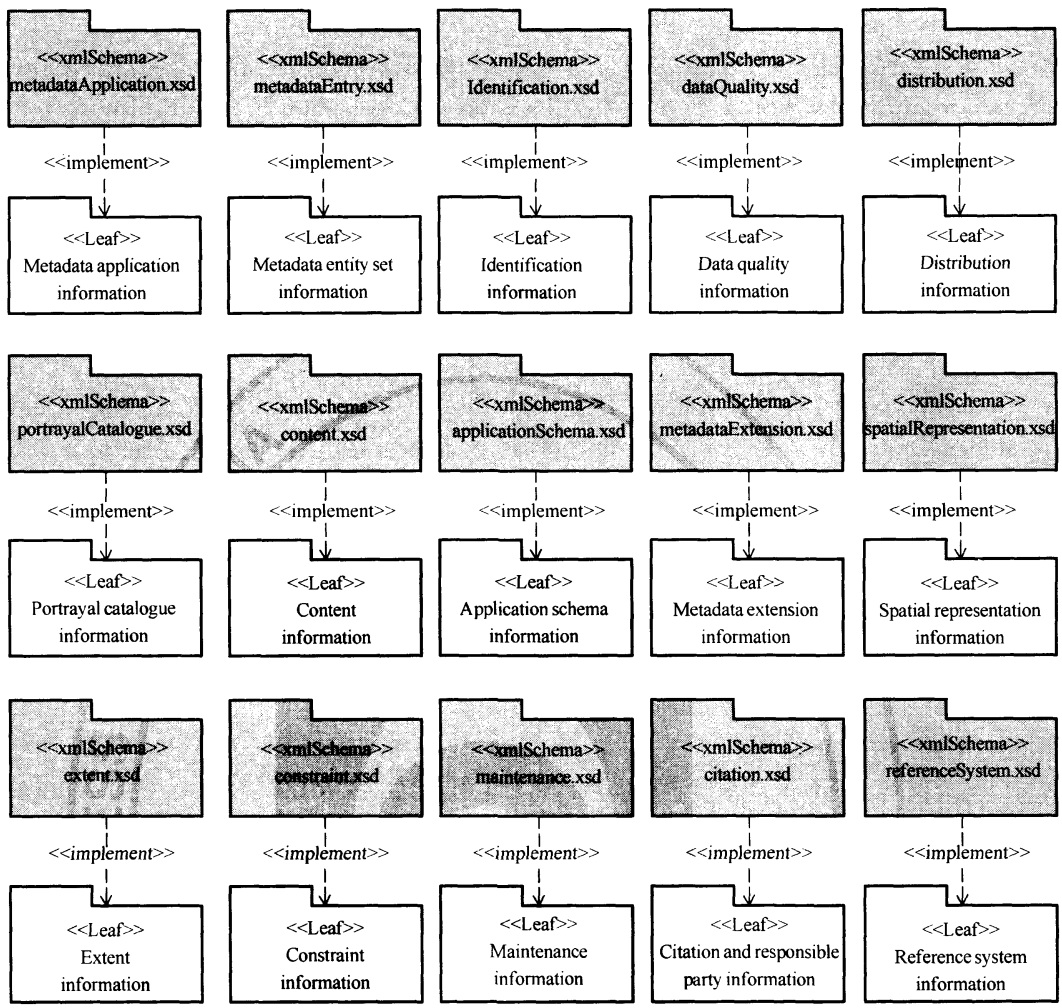

图 39（续）

9.3.4 metadataEntity. xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.2.1 中定义的 UML 概念模式，包括 MD_Metadata 类的实现。

本指导性技术文件的 MD_Metadata 类遵循第 8 章的编码规则。

9.3.5 identification. xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.2.2 中定义的 UML 概念模式，实现的类包括：

MD_Identification	MD_BrowseGraphic	MD_DataIdentification
MD_RepresentativeFraction	MD_Usage	MD_Keywords
DS_Association	MD_AggregateInformation	MD_CharacterSetCode
MD_SpatialRepresentationTypeCode	MD_ServiceIdentification	MD_TopicCategoryCode
MD_ProgressCode	MD_KeywordTypeCode	DS_AssociationTypeCode
DS_InitiativeTypeCode	MD_ResolutionType	

该 XML 模式中实现的类遵循第 8 章的编码规则。

9.3.6 constraints. xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.2.3 中定义的 UML 概念模式，实现的类包括：MD_Constraints，MD_LegalConstraints，MD_SecurityConstraints，MD_ClassificationCode，MD_RestrictionCode。

该 XML 模式中实现的类遵循第 8 章的编码规则。

9.3.7 dataQuality.xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.2.4 中定义的 UML 概念模式，实现的类包括：

DQ_ThematicClassificationCorrectness	DQ_TopologicalConsistency	LI_ProcessStep
DQ_AbsoluteExternalPositionalAccuracy	DQ_ConformanceResult	LI_Source
DQ_RelativeInternalPositionalAccuracy	DQ_QuantitativeResult	LI_Lineage
DQ_GriddedDataPositionalAccuracy	DQ_TemporalAccuracy	DQ_Result
DQ_NonQuantitativeAttributeAccuracy	DQ_DomainConsistency	DQ_DataQuality
DQ_AccuracyOfATimeMeasurement	DQ_ThematicAccuracy	DQ_Element
DQ_QuantitativeAttributeAccuracy	DQ_ConceptualConsistency	DQ_TemporalValidity
DQ_CompletenessCommission	DQ_LogicalConsistency	DQ_TemporalValidity
DQ_CompletenessOmission	DQ_PositionalAccuracy	DQ_Completeness
DQ_FormatConsistency		

该 XML 模式中实现的类遵循第 8 章的编码规则。

9.3.8 maintenance.xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.2.5 中定义的 UML 概念模式，实现的类包括：MD_MaintenanceInformation，MD_MaintenanceFrequencyCode，MD_ScopeCode，MD_ScopeDescription。

该 XML 模式中实现的类遵循第 8 章的编码规则。

9.3.9 spatialRepresentation.xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.2.6 中定义的 UML 概念模式，实现的类包括：

MD_GridSpatialRepresentation	MD_SpatialRepresentation	MD_Dimension
MD_VectorSpatialRepresentation	MD_TopologyLevelCode	MD_Georectified
MD_GeometricObjectTypeCode	MD_CellGeometryCode	MD_Georeferenceable
MD_DimensionNameTypeCode	MD_PixelOrientationCode	MD_GeometricObjects

该 XML 模式中实现的类遵循第 8 章的编码规则。

9.3.10 referenceSystem.xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.2.7 中定义的 UML 概念模式，实现的类包括：RS_Identifier，MD_ReferenceSystem 和 RS_Reference System。

该 XML 模式中实现的类遵循第 8 章的编码规则。

9.3.11 content.xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.2.8 中定义的 UML 概念模式，实现的类包括：MD_FeatureCatalogueDescription，MD_CoverageDescription，MD_ImageDescription，MD_ContentInformation，MD_RangeDimension，MD_Band，MD_CoverageContentTypeCode，MD_ImagingConditionCode。

该 XML 模式中实现的类遵循第 8 章的编码规则。

9.3.12 portrayalCatalogue.xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.2.9 中定义的 UML 概念模式，实现的类包括：MD_PortrayalCatalogueReference。

该 XML 模式中实现的类遵循第 8 章的编码规则。

9.3.13 distribution.xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.2.10 中定义的 UML 概念模式，实现的类包括：MD_Medium，MD_DigitalTransferOptions，MD_StandardOrderProcess，MD_Distributor，MD_Distribution，MD_Format，MD_MediumFormatCode，MD_MediumNameCode。

该 XML 模式中实现的类遵循第 8 章的编码规则。

9.3.14 metadataExtension.xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.2.11 中定义的 UML 概念模式，实现的类包括：MD_ExtendedElementInformation，MD_MetadataExtensionInformation，MD_ObligationCode，MD_DatatypeCode。

该 XML 模式中实现的类遵循第 8 章的编码规则。

9.3.15 applicationSchema.xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.2.12 中定义的 UML 概念模式，实现的类包括：MD_ApplicationSchemaInformation。

该 XML 模式中实现的类遵循第 8 章的编码规则。

9.3.16 extent.xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.3.1 中定义的 UML 概念模式，实现的类包括：EX_TemporalExtent，EX_VerticalExtent，EX_BoundingPolygon，EX_Extent，EX_GeographicExtent，EX_GeographicBoundingBox，EX_SpatialTemporalExtent，EX_GeographicDescription。

该 XML 模式中实现的类遵循第 8 章的编码规则。

9.3.17 citation.xsd

该 XML 模式实现了 GB/T 19710—2005 的 A.3.2 中定义的 UML 概念模式，实现的类包括：CI_ResponsibleParty，CI_Citation，CI_Address，CI_OnlineResource，CI_Contact，CI_Telephone，URL，CI_Date，CI_Series，CI_RoleCode，CI_PresentationFormCode，CI_OnLineFunctionCode，CI_DateTypeCode。

该 XML 模式中实现的类遵循第 8 章的编码规则，但 URL 类除外，如图 40 所示，该类实现为一个映射到 xs:anyURI 的 XML 简单类型。

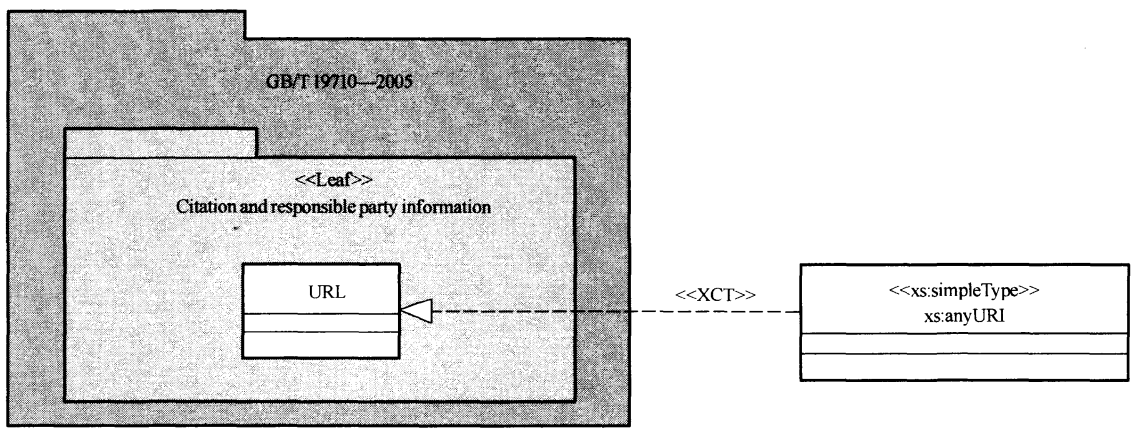

图 40 URL 的实现

9.4 gss 命名空间

9.4.1 gss 命名空间的构成

GB/T 19710—2005 的实现需要对 GB/T 23707—2009 中几个元素进行编码。由于这些元素不是专门针对地理信息元数据的，单独建立了一个包含 GB/T 23707—2009 中的元素的 XML 模式的命名空间，即 http://www.isotc211.org/2005/gss，用来表示该命名空间的通用前缀为 gss，表示地理空间模式。该命名空间的根为 gss.xsd，gss 命名空间的构成如图 41 所示。

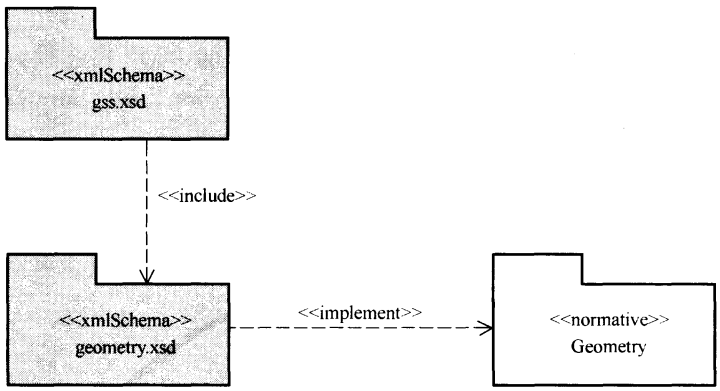

图 41 gss 命名空间的构成

9.4.2 gss.xsd

该 XML 模式直接或间接包含 gss 命名空间中实现的概念，但不包含任何类型的申明。

9.4.3 geometry.xsd

该 XML 模式包含 GM_Object 和 GM_Point 的实现，这些类的编码被映射到 GB/T 23708—2009 的几何类型，图 42 表示了这种映射关系。

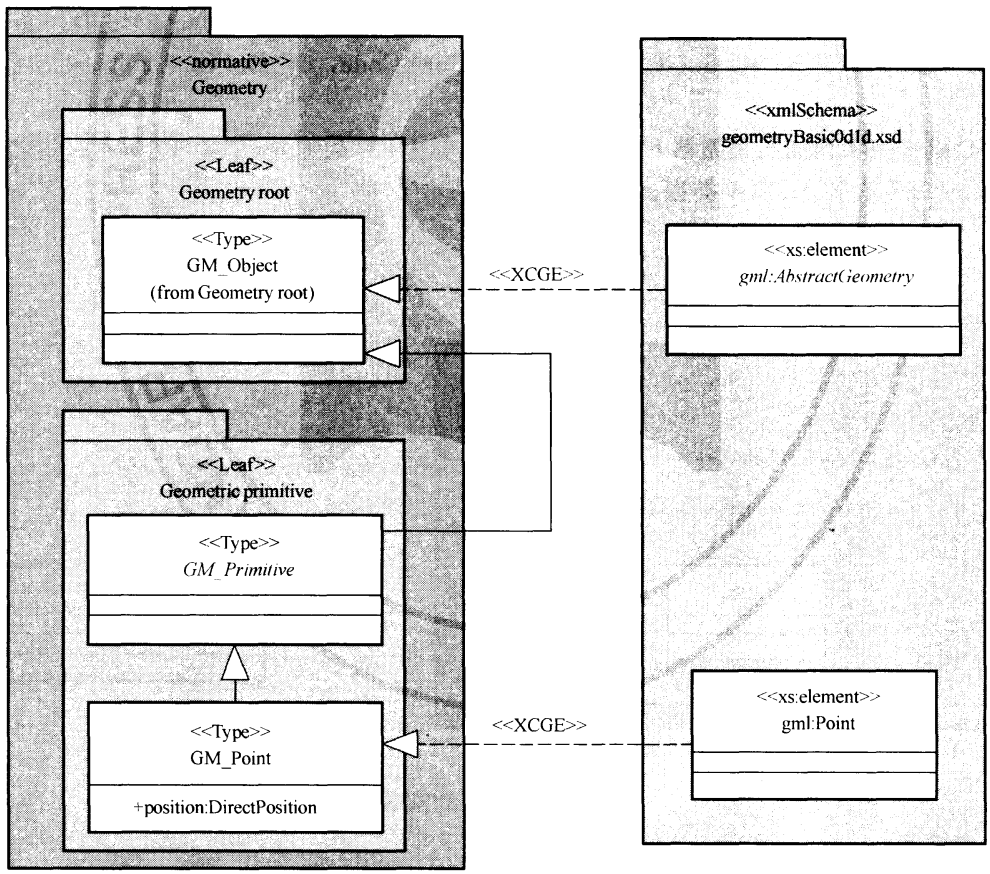

图 42 GB/T 23707—2009 到 GB/T 23708—2009 的映射

9.5 gts 命名空间

9.5.1 gts 命名空间的构成

GB/T 19710—2005 的实现需要对 GB/T 22022—2008 中几个元素进行编码。由于这些元素不是专门针对地理信息元数据的，单独建立了一个包含 GB/T 22022—2008 中的元素的 XML 模式的命名空间，即 http://www.isotc211.org/2005/gts，用来表示该命名空间的通用前缀为 gts，表示地理时间模