INTERNATIONAL STANDARD ISO/IEC 19503

First edition
2005-11-01

Information technology — XML Metadata Interchange (XMI)

Technologies de l'information — Échange de métadonnées XML (XMI)

Reference number
ISO/IEC 19503:2005(E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 19503 was prepared by the Object Mangement Group (OMG) and was adopted, under the PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

ISO/IEC 19503 is related to

— ISO/IEC 19501, *Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2*

— ISO/IEC 19502, *Information technology — Meta Object Facility (MOF)*

Introduction

The main purpose of this International Standard (XML) is to enable easy interchange of metadata between application development lifecycle tools (such as modeling tools based on the Unified Modeling Language (UML), ISO/IEC 19501, and metadata repositories/frameworks based on the Meta Object Facility (MOF), ISO/IEC 19502) in distributed heterogeneous environments. This International Standard integrates three key industry standards:

- XML - eXtensible Markup Language, a W3C standard.
- UML - Unified Modeling Language, an OMG modeling specification, which is now ISO/IEC 19501.
- MOF - Meta Object Facility (ISO/IEC 19502).

The OMG adopted the XMI (version 1.0) in February 1999. It was developed as a response to a request for proposal, issued by the OMG Analysis and Design Task Force, for a model and metadata interchange facility. The purpose of the facility was to support the interchange of metadata (such as ODP UML models). The most recent revision of XMI, 2.0, was submitted by the XMI Revision Task Force in October, 2002, and includes corrections and clarifications to the original specification, and changes to accommodate revisions to the 1.4 version of MOF.

The rapid growth of distributed processing has led to a need for a coordinating framework for this standardization and ITU-T Recommendations X.901-904 | ISO/IEC 10746, *Open Distributed Processing — Reference Model* (RM-ODP) provides such a framework. It defines an architecture in which support of distribution, interoperability, and portability can be integrated. RM-ODP Part 2 (ISO/IEC 10746-2) defines the foundational concepts and modeling framework for describing distributed systems. RM-ODP Part 3 (ISO/IEC 10746-3) specifies a generic architecture of open distributed systems, expressed using the foundational concepts and framework defined in Part 2.

While not limited to this context, this International Standard is relevant to work on the standardization of Open Distributed Processing (ODP).

Information technology — XML Metadata Interchange (XMI)

1 Scope

This International Standard provides specifications for:

a. A set of XML Schema Definitions (XSD) production rules for transforming MOF based metamodels into XML Schemas.

b. A set of XML Document production rules for encoding and decoding MOF based metadata.

c. Design principles for XMI based Schemas and XML documents.

d. A set of production rules for importing XML DTDs to a MOF based metamodel.

This International Standard enhances metadata management and metadata interoperability in distributed object environments in general and in distributed development environments in particular. While this International Standard addresses stream based metadata interoperability in the object analysis and design domain, XMI (in part because it is MOF based) is equally applicable to metadata in many other domains.

2 Normative references

The following referenced documents are indispensable for the application 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.

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation X.902 (1996) | ISO/IEC 10746-2:1996, *Information technology — Open Distributed Processing — Reference Model: Foundations*
- ITU-T Recommendation X.903 (1996) | ISO/IEC 10746-3:1996, *Information technology — Open Distributed Processing — Reference Model: Architecture*

2.2 International Standards

- ISO/IEC 10646:2003, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*
- ISO/IEC 19501, *Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2*
- ISO/IEC 19502, *Information technology — Meta Object Facility (MOF)*
- W3C XML 1.0 : http://www.w3.org/TR/REC-xml – February, 2004
- W3C XSD 1.0 http://www.w3.org/TR/xmlschema-0/, xmlschema-1, xmlschema-2

3 Abbreviations

DTD	Document Type Definition
MOF	Meta Object Facility
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XSD	XML Schema Definition

4 XMI Schema Design Principles

4.1 Purpose

This Clause contains a description of the XML Schemas that may be used with the XMI specification to allow some metamodel information to be verified through XML validation. The use of schemas in XMI is described first, followed by a brief description of some basic principles, which includes a short description of each XML attribute and XML element defined by XMI. Those descriptions are followed by more complete descriptions that provide examples illustrating the motivation for the XMI schema design in the areas of metamodel class specification, transmitting incomplete metadata, linking, tailoring schema production, transmitting metadata differences, and exchanging documents between tools.

It is possible to define how to automatically generate a schema from the MOF metamodel to represent any MOF-compliant metamodel. That definition is presented in Clause 5.

This Clause describes XMI 2.0 schemas; Clause 5 describes how to create XMI 2.0 schemas.

You may specify tag value pairs as part of the MOF metamodel to tailor the schemas that are generated, but you are not required to do so. Using these tag value pairs requires some knowledge of XML schemas, but the schemas that are produced might perform more validation than the default schemas. See Clause 7 for a complete description of how to generate XML schemas using these tag value pairs. Sub clause 4.11, “Tailoring Schema Production,” on page 23 describes the tag values, their affect on schema production, and their impact on document serialization.

4.2 Use of XML Schemas

An XML schema provides a means by which an XML processor can validate the syntax and some of the semantics of an XML document. This specification provides rules by which a schema can be generated for any valid XMI-transmissible MOF-based metamodel. However, the use of schemas is optional; an XML document need not reference a schema, even if one exists. The resulting document can be processed more quickly, at the cost of some loss of confidence in the quality of the document.

It can be advantageous to perform XML validation on the XML document containing MOF metamodel data. If XML validation is performed, any XML processor can perform some verification, relieving import/export programs of the burden of performing these checks. It is expected that the software program that performs verification will not be able to rely solely on XML validation for all of the verification, however, since XML validation does not perform all of the verification that could be done.

Each XML document that contains metamodel data conforming to this specification contains: XML elements that are required by this specification, XML elements that contain data that conform to a metamodel, and, optionally, XML elements that contain metadata that represent extensions of the metamodel. Metamodels are explicitly identified in XML elements required by this specification. Some metamodel information can also be encoded in an XML schema. Performing XML validation provides useful checking of the XML elements that contain metadata about the information transferred, the transfer information itself, and any extensions to the metamodel.

The XML Namespace specification has been adopted by the W3C, allowing XMI to use multiple metamodels at the same time. XML schema validation works with XML namespaces, so you can choose your own namespace prefixes in an XML document and use a schema to validate it. The namespace URIs, not the namespace prefixes, are used to identify which schemas to use to validate an XML document.

4.2.1 XML Validation of XMI documents

XML validation can determine whether the XML elements required by this specification are present in the XML document containing metamodel data, whether XML attributes that are required in these XML elements have values for them, and whether some of the values are correct.

XML validation can also perform some verification that the metamodel data conforms to a metamodel. Although some checking can be done, it is impossible to rely solely on XML validation to verify that the information transferred satisfies all of a metamodel's semantic constraints. Complete verification cannot be done through XML validation because it is not currently possible to specify all of the semantic constraints for a metamodel in an XML schema, and the rules for automatic generation of a schema preclude the use of semantic constraints that could be encoded in a schema manually, but cannot be automatically encoded.

Finally, XML validation can be used to validate extensions to the metamodel, because extensions must be represented as elements; if those elements are defined in a schema, the schema can be used to verify the elements.

4.2.2 Requirements for XMI Schemas

Each schema used by XMI must satisfy the following requirements:

- All XML elements and attributes defined by the XMI specification must be imported in the schema. They cannot be put directly in the schema itself, since there is only one target namespace per schema.
- Metamodel constructs have corresponding element declarations, and may have an XML attribute declaration, as described below. In addition, some constructs also have a complexType declaration. The declarations may utilize groups, attribute groups, and types, as described below.
- Any XML elements that represent extensions to the metamodel may be declared in a schema, although it is not necessary to do so.

By default, XMI schemas allow incomplete metadata to be transmitted, but you can enforce the lower bound of multiplicities if you wish. See 4.9, "Transmitting Incomplete Metadata," on page 18 for further details.

4.3 Basic Principles

This sub clause discusses the basic organization of an XML schema for XMI. Detailed information about each of these topics is included later in this Clause.

4.3.1 Required XML Declarations

This specification requires that XML element declarations, types, attributes, and attribute groups be included in schemas to enable XML validation of metadata that conforms to this specification. Some of these XML elements contain metadata about the metadata to be transferred. For example, the identity of the metamodel associated with the metadata, the tool that generated the metadata, whether the metadata has been verified, etc.

All XML elements defined by this specification are in the namespace "http://www.omg.org/XMI." The XML namespace mechanism can be used to avoid name conflicts between the XMI elements and the XML elements from your MOF models.

In addition to required XML element declarations, there are some attributes that must be defined according to this specification. Every XML element that corresponds to a metamodel class must have XML attributes that enable the XML element to act as a proxy for a local or remote XML element. These attributes are used to associate an XML element with another XML element. There are also other required attributes to let you put data in XML attributes rather than XML elements. You may customize the declarations using MOF tag values.

4.3.2 Metamodel Class Representation

Every metamodel class is represented in the schema by an XML element whose name is the class name, as well as a complexType whose name is the class name. The declaration of the type lists the attributes of the class; references to association ends relating to the class; and the classes that this class contains, either explicitly or through composition associations. By default, the content models of XML elements corresponding to metamodel classes do not impose an order on the attributes and references.

By default, XMI allows you to serialize features using either XML elements or XML attributes; however, XMI allows you to specify how to serialize them if you wish. Containment references and multivalued attributes always are serialized using XML elements.

4.3.3 Metamodel Extension Mechanism

Every XMI schema contains a mechanism for extending a metamodel class. Zero or more **extension** elements are included in the content model of each class. These extension elements have a content model of ANY, allowing considerable freedom in the nature of the extensions. The processContents attribute is lax, which means that processors will validate the elements in the extension if a schema is available for them, but will not report an error if there is no schema for them. In addition, the top level XMI element may contain zero or more **extension** elements, which provides for the inclusion of any new information. One use of the extension mechanism might be to associate display information for a particular tool with the metamodel class represented by the XML element. Another use might be to transmit data that represents extensions to a metamodel.

Tools that rely on XMI are expected to store the extension information and export it again to enable round trip engineering, even though it is unlikely they will be able to process it further. XML elements that are put in the **extension** elements may be declared in schemas, but are not required to be.

4.4 XMI Schema and Document Structure

Every XMI schema consists of the following declarations:

- An XML version processing instruction. Example: <? XML version=”1.0” ?>
- An optional encoding declaration that specifies the character set, which follows the ISO-10646 (also called extended Unicode) standard.
  Example: <? XML version=”1.0” ENCODING=”UCS-2” ?>.
- Any other valid XML processing instructions.
- A schema XML element.
- An import XML element for the XMI namespace.
- Declarations for a specific metamodel.

Every XMI document consists of the following declarations, unless the XMI is embedded in another XML document:

- An XML version processing instruction.
- An optional encoding declaration that specifies the character set.
- Any other valid XML processing instructions.

XMI imposes no ordering requirements beyond those defined by XML. XML Namespaces may also be declared in the XMI element as described below.

The top element of the XMI information structure is either the XMI element, or an XML element corresponding to an instance of a class in the MOF metamodel. An XML document containing only XMI information will have XMI as the root element of the document. It is possible for future XML exchange formats to be developed that extend XMI and embed XMI elements within their XML elements.

4.5 XMI Model

This sub clause describes the model for XMI document structure, called the XMI model. The XMI model is an instance of MOF for describing the XMI-specific information in an XMI document, such as the version, documentation, extensions, and differences.

Using an XMI model enables XMI document metadata to be treated in the same fashion as other MOF metadata, allowing use of standard MOF APIs for access to and construction of XMI-specific information in the same manner as other MOF objects. A valid XMI document may contain XMI metadata but is not required to.

4.5.1 XML Schema for the XMI Model

When the XMI model is generated as an XML Schema following the XMI schema production rules, the result is a set of XML element and attribute declarations. These declarations are shown in Clause 8 and given the XML namespace "http://www.omg.org/XMI." Every XMI-compliant schema must include the declarations of the following XML elements by importing the declarations in the XMI namespace "http://www.omg.org/XMI."

In addition, there are attribute declarations and attributeGroup declarations that must be imported also. These include the id attribute, and the IdentityAttribs, LinkAttribs, and ObjectAttribs attribute groups. These constructs are not defined in the XMI model.

In the declarations that follow, the XML Schema namespace, whose URI is "http://www.w3.org/2001/XMLSchema," has the namespace prefix "xsd;" the XMI namespace is the default namespace.

4.5.2 XMI Model Classes

There are three diagrams that describe the XMI model. The details of the classes are described in the sub clauses below. This sub clause gives an overview of the model.

Figure 4.1 shows the XMI element, documentation, and extension elements. The XMI class is an overall default container for XMI document metadata and contents. The attributes of the XMI class are the version, documentations, differences (add, replace, delete in Figure 4.2), and extensions. The Documentation class contains many fields to describe the document for non-computational purposes. The Extension class contains the metadata for external information. The String datatype is the data type for strings in the MOF model with XML Schema data type of "http://www.w3.org/2001/XMLSchema#string." The Integer datatype is the data type for integers in the MOF model with XML Schema data type of "http://www.w3.org/2001/XMLSchema#integer."

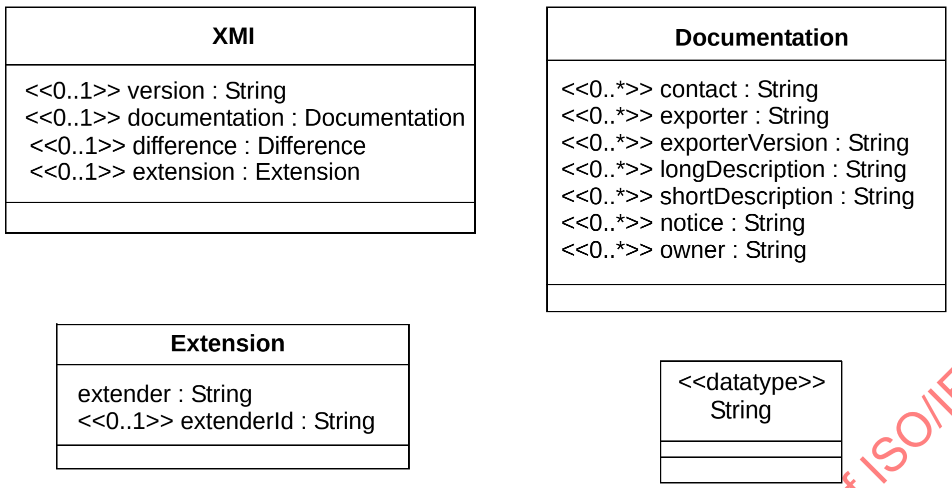

XMI
<<0..1>> version : String
<<0..1>> documentation : Documentation
<<0..1>> difference : Difference
<<0..1>> extension : Extension
Documentation
<<0..*>> contact : String
<<0..*>> exporter : String
<<0..*>> exporterVersion : String
<<0..*>> longDescription : String
<<0..*>> shortDescription : String
<<0..*>> notice : String
<<0..*>> owner : String
Extension
extender : String
<<0..1>> extenderId : String
<<datatype>>
String

Figure 4.1 - The XMI Model for the XMI element, documentation, and extension

The differences information (Figure 4.2) is described as additions, deletions, and replacements to target objects. The objects referenced by the differences may be in the same or different documents. The differences information consists of the Add, Delete, and Replace classes, which specify a set of differences and refer to MOF objects that are added or removed. Note that the RefBaseObject class is a placeholder for specifying that a Difference has a target that can refer to any objects. The RefObject class is not included in the required element declarations.

The XML Schema declarations for each element of the XML model are given in the following sub clauses. They may be generated by following the XMI production of XML Schema rules defined in Clause 7, except for the XMI class and the XMI attributes described in 4.6, “XMI Attributes,” on page 12.

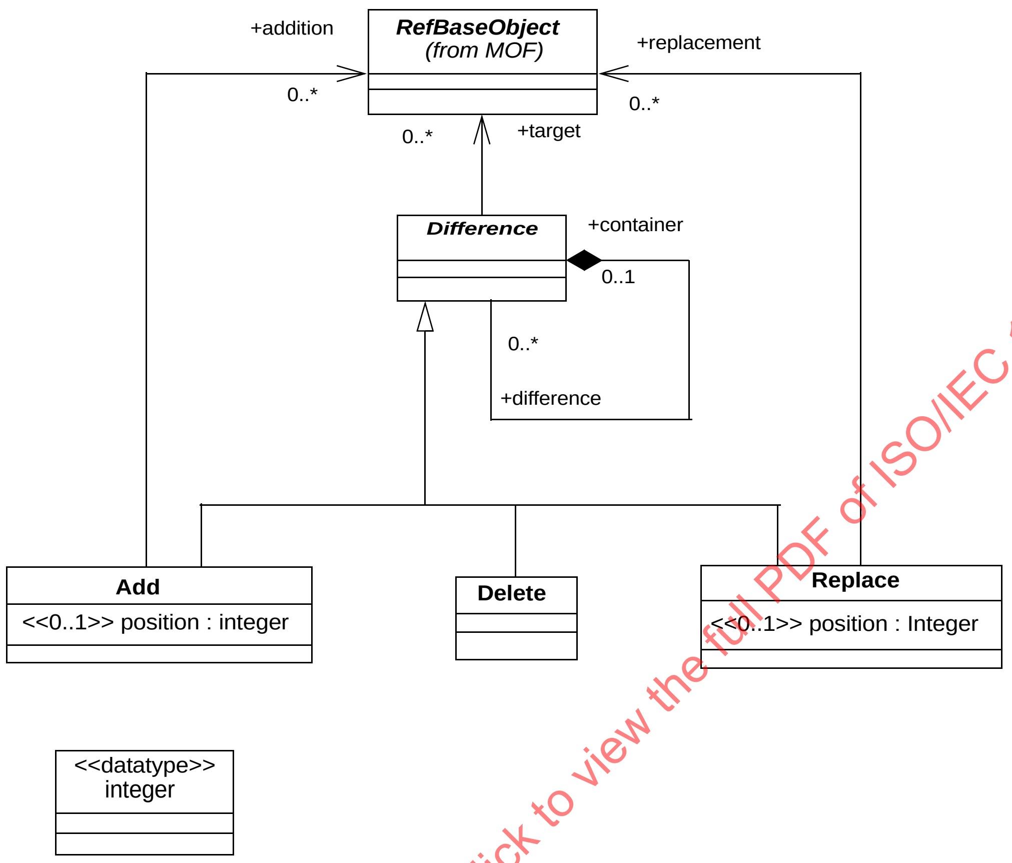


Figure 4.2 - The XMI Model for differences

4.5.3 XMI

The top level XML element for XMI documents containing only XMI data is the XMI element. Its declaration is:

```
<xsd:complexType name="XMI">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:any processContents="strict"/>
  </xsd:choice>
  <xsd:attribute ref="id"/>
  <xsd:attributeGroup ref="IdentityAttribs"/>
  <xsd:attributeGroup ref="LinkAttribs"/>
  <xsd:attribute name="type" type="xsd:QName" use="optional"
                     form="qualified"/>
  <xsd:attribute name="version" type="xsd:string" use="required" fixed="2.0"
                 form="qualified"/>
</xsd:complexType>

<xsd:element name="XMI" type="XMI"/>
```

The **version** attribute is required to be set to “2.0.” This indicates that the metadata conforms to this version of the XMI specification. Revised versions of this standard will have another number assigned by the OMG.

The XMI element need not be the root element of an XML document; you can include it inside any XML element that was not serialized according to this specification. If a document contains only XMI information, the XMI element is typically not present when there is only a single top-level object. The xmi:version attribute is used to denote the start of XMI information and identify the XMI version, when the XMI element itself is not present. Clause 8 contains examples of the use of the XMI element.

The XMI class has the tag contentType set to “any” to indicate that any XMI element may be present in the XMI stream.

The attribute version has the tag form set to “qualified,” the tag fixedValue set to “2.0,” the tag attribute set to “true,” and the tag enforceMinimumMultiplicity set to “true.”

Since the XMI model is an instance of MOF, it can be serialized using the same rules as any other MOF metamodel, with one exception. Using the default serialization rules would result in the XMI version attribute appearing twice in XMI elements: once directly from the XMI version attribute, and once through the inclusion of the ObjectAttribs group. Therefore, the version attribute that belongs to the ObjectAttribs attribute group must be excluded from the XMI type declaration. See 6.3.1, “Overall Document Structure,” on page 50 for details on how the XMI class is serialized.

The serialization of the XMI element is special — it is defined by the XML Document Production rules in Clause 8.

The XMI model package has the following tag settings:

- tag nsURI set to “http://www.omg.org/XMI”
- tag nsPrefix set to “xmi”
- tag superClassFirst set to “true”
- tag useSchemaExtension set to “true”

4.5.4 Extension

The Extension class is designed to contain extended information outside the scope of the user metamodel. Extensions are a multivalued attribute of the XMI class and may also be embedded in specific locations in an XMI document. The Schema for extension is:

```
<xsd:complexType name="Extension">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:any processContents="lax"/>
  </xsd:choice>
  <xsd:attribute ref="id"/>
  <xsd:attributeGroup ref="ObjectAttribs"/>
  <xsd:attribute name="extender" type="xsd:string" use="optional"/>
  <xsd:attribute name="extenderID" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:element name="Extension" type="Extension"/>
```

The **extender** attribute should indicate which tool made the extension. It is provided so that tools may ignore the extensions made by other tools before the content of the extensions element is processed. The **extenderID** is an optional internal ID from the extending tool. The other attributes allow individual extensions to be identified and to act as proxies for local or remote extensions.

The Extension class in the MOF model has the tag contentType set to “any” and the processContents tag set to “lax.” The extender and extenderID attributes have the tag attribute set to “true.”

4.5.5 Documentation

The Documentation class contains information about the XMI document or stream being transmitted, for instance the owner of the document, a contact person for the document, long and short descriptions of the document, the exporter tool that created the document, the version of the tool, and copyright or other legal notices regarding the document. The data type of all the attributes of Documentation is string. The XML Schema generated for Documentation is:

```
<xsd:complexType name="Documentation">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="contact" type="xsd:string"/>
    <xsd:element name="exporter" type="xsd:string"/>
    <xsd:element name="exporterVersion" type="xsd:string"/>
    <xsd:element name="longDescription" type="xsd:string"/>
    <xsd:element name="shortDescription" type="xsd:string"/>
    <xsd:element name="notice" type="xsd:string"/>
    <xsd:element name="owner" type="xsd:string"/>
    <xsd:element ref="Extension"/>
  </xsd:choice>
  <xsd:attribute ref="id"/>
  <xsd:attributeGroup ref="ObjectAttribs"/>
  <xsd:attribute name="contact" type="xsd:string" use="optional"/>
  <xsd:attribute name="exporter" type="xsd:string" use="optional"/>
  <xsd:attribute name="exporterVersion" type="xsd:string" use="optional"/>
  <xsd:attribute name="longDescription" type="xsd:string" use="optional"/>
  <xsd:attribute name="shortDescription" type="xsd:string" use="optional"/>
  <xsd:attribute name="notice" type="xsd:string" use="optional"/>
  <xsd:attribute name="owner" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:element name="Documentation" type="Documentation"/>
```

4.5.6 Add, Replace, and Delete

The Add class represents an addition to a target set of objects in this document or other documents. The **position** attribute indicates where to place the addition relative to other XML elements. The default, -1, indicates to add the new elements at the end of the target element. The **addition** attribute refers to the set of objects to be added. Both of these attributes have the tag attribute set to “true.”

The Replace class represents the deletion of the target set of objects and the addition of the objects referred to in the replacement attribute. The **position** attribute indicates where to place the replacement relative to other XML elements. The default, -1, indicates to add the replacing elements at the end of the target element. The **replacement** attribute refers to the object that will replace the target element. Both of these attributes have the tag attribute set to “true.”

The Delete class represents a deletion to a target set of objects in this document or other documents.

The Difference class is the superclass for the Add, Replace, and Delete classes.

The declarations for these classes are:

```
<xsd:complexType name="Difference">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="target">
      <xsd:complexType>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:any processContents="skip"/>
        </xsd:choice>
        <xsd:anyAttribute processContents="skip"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="difference" type="Difference"/>
    <xsd:element name="container" type="Difference"/>
    <xsd:element ref="Extension"/>
  </xsd:choice>
  <xsd:attribute ref="id"/>
  <xsd:attributeGroup ref="ObjectAttribs"/>
  <xsd:attribute name="target" type="xsd:IDREFS" use="optional"/>
  <xsd:attribute name="container" type="xsd:IDREFS" use="optional"/>
</xsd:complexType>

<xsd:element name="Difference" type="Difference"/>

<xsd:complexType name="Add">
  <xsd:complexContent>
    <xsd:extension base="Difference">
      <xsd:attribute name="position" type="xsd:string" use="optional"/>
      <xsd:attribute name="addition" type="xsd:IDREFS" use="optional"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Add" type="Add"/>

<xsd:complexType name="Replace">
  <xsd:complexContent>
    <xsd:extension base="Difference">
      <xsd:attribute name="position" type="xsd:string" use="optional"/>
      <xsd:attribute name="replacement" type="xsd:IDREFS" use="optional"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Replace" type="Replace"/>
```

```
<xsd:complexType name="Delete">
  <xsd:complexContent>
    <xsd:extension base="Difference"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Delete" type="Delete"/>
```

4.6 XMI Attributes

This sub clause describes the XML attributes that are used in the production of XML documents and Schemas. By defining a consistent set of XML attributes, XMI provides a consistent architectural structure enabling consistent object identity and linking across all assets.

4.6.1 Element Identification Attributes

Three XML attributes are defined by this specification to identify XML elements so that XML elements can be associated with each other. The purpose of these attributes is to allow XML elements to reference other XML elements using XML IDREFs, XLinks, and XPointers.

Two of these attributes are declared in an attribute group called **IdentityAttribs**; the id attribute is declared globally, because you may change the name of the id attribute using the idName tag. Placing these attributes in an attribute group prevents errors in the declarations of these attributes in schemas. Its declaration is as follows:

```
<xsd:attribute name="id"  type="xsd:ID"/>

<xsd:attributeGroup name="IdentityAttribs">
   <xsd:attribute name="label" type="xsd:string" use="optional"
                           form="qualified"/>
   <xsd:attribute name="uuid" type="xsd:string" use="optional"
                            form="qualified"/>
</xsd:attributeGroup>
```

id

XML semantics require the values of this attribute to be unique within an XML document; however, the value is not required to be globally unique. This attribute may be used as the value of the **idref** attribute defined in the next sub clause. It may also be included as part of the value of the **href** attribute in XLinks. An example of the use of this attribute and the other attributes in this sub clause can be found in 4.10.3, “Example from UML,” on page 22.

label

This attribute may be used to provide a string label identifying a particular XML element. Users may put any value in this attribute.

uuid

The purpose of this attribute is to provide a globally unique identifier for an XML element. The values of this attribute should be globally unique strings prefixed by the type of identifier. If you have access to the UUID assigned in MOF, you may put the MOF UUID in the uuid XML attribute when encoding the MOF data in XMI. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by “**DCE**:”. The values of this attribute may be used in the **href** attribute in simple XLinks. XMI does not specify which UUID convention is chosen.

The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed.

When a UUID is placed in an XMI file, the form is “id namespace:uuid.” The id namespace of UUIDs is typically DCE. An example is “DCE:2fac1234-31f8-11b4-a222-08002b34c003.”

The MOF refID() is often used as the uuid in XMI implementations.

4.6.2 Linking Attributes

XMI requires the use of several XML attributes to enable XML elements to refer to other XML elements using the values of the attributes defined in the previous sub clause. The purpose of these attributes is to allow XML elements to act as simple XLinks or to hold a reference to an XML element in the same document using the XML IDREF mechanism. See 4.10, “Linking,” on page 19.

The attributes described in this sub clause are included in an attribute group called **LinkAttribs**. The attribute group declaration is:

```
<xsd:attributeGroup name="LinkAttribs">
  <xsd:attribute name="href" type="xsd:string" use="optional"/>
  <xsd:attribute name="idref" type="xsd:IDREF" use="optional"
                    form="qualified"/>
</xsd:attributeGroup>
```

The link attributes act as a union of two linking mechanisms, any one of which may be used at one time. The mechanisms are the XLink **href** for advanced linking across or within a document, or the **idref** for linking within a document.

Simple XLink Attributes

The **href** attribute declared in the above entity enables an XML element to act in a fashion compatible with the simple XLink according to the XLink and XPointer W3C recommendations. The declaration and use of **href** is defined in the XLink and XPointer specifications. XMI enables the use of simple XLinks. XMI does not preclude the use of extended XLinks. The XLink specification defines many additional XML attributes, and it is permissible to use them in addition to the attributes defined in the LinkAttribs group.

To use simple XLinks, set **href** to the URI of the desired location. The **href** attribute can be used to reference XML elements whose **id** attributes are set to particular values. The **id** attribute value can be specified using a special URI form for XPointers defined in the XLink and XPointer recommendations.

idref

This attribute allows an XML element to refer to another XML element within the same document using the XML IDREF mechanism. In XMI documents, the value of this attribute should be the value of the **id** attribute of the XML element being referenced.

4.6.3 Version Attribute

The version attribute must be present for XMI objects that are not serialized in an XMI XML element and are not serialized in an XML element representing another object. The attribute value, if present, must be 2.0, indicating that the object was serialized according to this specification:

```
<xsd:attribute name="version" type="xsd:string" fixed="2.0"/>
```

4.6.4 Type Attribute

The type attribute is used to specify the type of object being serialized, when the type is not known from the model. This can occur if the type of a reference has subclasses, for instance. The declaration of the attribute is:

```
<xsd:attribute name="type" type="xsd:QName" form="qualified"/>
```

Rather than including the IdentityAttribs, and LinkAttribs attribute groups, and the version and type attributes in the declarations for each MOF class, the XMI namespace includes the following declaration of the **ObjectAttribs** attribute group for the attribute declarations that pertain to objects:

```
<xsd:attributeGroup name="ObjectAttribs">
  <xsd:attributeGroup ref="IdentityAttribs"/>
  <xsd:attributeGroup ref="LinkAttribs"/>
  <xsd:attribute name="version" type="xsd:string" use="optional" fixed="2.0"
                    form="qualified"/>
  <xsd:attribute name="type" type="xsd:QName" use="optional"
                    form="qualified"/>
</xsd:attributeGroup>
```

4.7 XMI Type

The XMI namespace contains a type called Any. It is used in the XMI 2.0 schema production rules for class attributes, class references, and class compositions. The declaration of this type is part of the fixed declarations for XMI 2.0. The Any type allows any content and any attributes to appear in elements of that type, skipping XML validation for the element’s content and attributes. The declaration of the type is as follows:

```
<xsd:complexType name="Any">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:any processContents="skip"/>
    </xsd:choice>
    <xsd:anyAttribute processContents="skip"/>
</xsd:complexType>
```

By using this type, the XMI 2.0 schema production rules generate smaller schemas than if this type was declared multiple times in a schema. Also, using the Any type enables some changes to be made to the Any type declaration without affecting generated XMI 2.0 schemas.

4.8 Metamodel Class Specification

This sub clause describes in detail how to represent information about metamodel classes in an XMI compliant schema. It uses the EBNF grammar in the "XML Schema Production" clause to describe the manner in which attributes, associations, and containment relationships are represented in an XML schema, including how inheritance between metamodel classes is handled. It uses a short example to explain the encoding.

4.8.1 Namespace Qualified XML Element Names

When the official schema for a metamodel is produced, the schema generator must choose one or more namespace URIs that uniquely identify the XML namespaces in the metamodel. XML processors will use those namespace URIs to identify the schemas to use for XML validation, as described in the XML schema specification.

The XML element name for each metamodel class, package, and association in a document is its short name. The name for XML tags corresponding to metamodel attributes and references is the short name of the attribute or reference. The name of XML attributes corresponding to metamodel references and metamodel attributes is the short name of the reference or attribute, since each tag in XML has its own namespace.

Each namespace is assigned a logical URI. The logical URI is placed in the namespace declaration of the XMI element in XML documents that contain instances of the metamodel. The XML namespace specification assigns logical names to namespaces that are expected to remain fixed throughout the life of all uses of the namespace since it provides a permanent global name for the resource. An example is "http://schema.omg.org/spec/UML/1.4." There is no requirement or expectation by the XML Namespace specification that the logical URI be resolved or dereferenced during processing of XML documents.

The following is an example of a UML model in an XMI document using namespaces.

```
<xmi:XMI version="2.0"
            xmlns:UML="http://schema.omg.org/spec/UML/1.4"
           xmlns:xmi="http://schema.omg.org/spec/XMI/2.0">
  <UML:Class name="C1">
    <feature xmi:type="UML:Attribute" name="a1" visibility="private"/>
  </UML:Class>
</xmi:XMI>
```

The model has a single class named C1 that contains a single attribute named a1 with visibility private. The XMI element declares the version of XMI and the namespace for UML with the logical URI.

4.8.2 Metamodel Multiplicities

In XMI 1.1, the multiplicities from the metamodel were ignored, since DTDs were not able to validate multiplicities without ordering the content of XML elements. By default, XMI produces schemas that ignore multiplicities also.

You may tailor the schemas produced by XMI by specifying tag values in the MOF metamodel. Two of the tags, "org.omg.xmi.enforceMaximumMultiplicity" and "org.omg.xmi.enforceMinimumMultiplicity" allow you to specify that multiplicities are to be used in a schema rather than being ignored.

Metamodel multiplicities map directly from the MOF definition of multiplicity, which is a lower bound and an upper bound, to schema XML attributes called "minOccurs" and "maxOccurs." The minOccurs XML attribute corresponds to the lower bound for the multiplicity, and the maxOccurs XML attribute corresponds to the upper bound for the multiplicity.

4.8.3 Class Specification

Every metamodel class is decomposed into three parts: attributes, associations, and compositions. A class is represented by an XML element, with an XML element for each attribute, reference, and composition. The XML element for the class includes the inherited attributes, associations, and composition.

In the examples that follow in this sub clause, “xsd” is the namespace prefix for the XML schema namespace (“http://www.w3.org/2001/XMLSchema”), and “xmi” is the namespace prefix for the XMI namespace.

The representation of a metamodel class named “c” is shown below for the simplest case where “c” does not have any attributes, associations, or containment relationships:

```
<xsd:element name="c" type="c"/>

<xsd:complexType name="c">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="xmi:Extension"/>
    </xsd:choice>
    <xsd:attribute ref="xmi:id"/>
    <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
</xsd:complexType>
```

If the class has attributes, associations, or compositions, the XML elements for them are put in the all group of the content model, as explained below.

4.8.4 Attribute Specification

The representation of attributes of metamodel class “c” uses XML elements and XML attributes. If the metamodel attribute types are primitives or enumerations, then by default XML attributes are declared for them as well as XML elements. The reasons for this encoding choice are several, including: the values to be exchanged may be very large values and unsuitable for XML attributes, and may have poor control of whitespace processing with options which apply only to element contents. The default encoding can be changed using the XMI “attribute” and “element” tags. See 4.11.4, “XML element vs XML attribute,” on page 25 for information on how these tags affect encoding. See 4.11.1, “XMI Tag Values,” on page 23 for a complete list of XMI tags.

The declaration of each attribute named “a” is as follows:

<xsd:element name="a" type="*type specification*"/>

The XML element corresponding to the attribute is declared in the content of the complexType corresponding to the class that owns the attribute. The type specification is either an XML schema data type, an enumeration data type, or a class from the metamodel.

For attributes whose types are string type and whose upper bound multiplicity is 1, an XML attribute must also be declared in the XML element corresponding to metamodel class “c,” and the XML element must be put in the content model of the XML element for class “c;” the declaration of “c” appears as follows without multiplicity enforcement:

```
<xsd:element name="c" type="c"/>

<xsd:complexType name="c">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="a" type="xsd:string" nillable="true"/>
      <xsd:element ref="xmi:Extension"/>
    </xsd:choice>
    <xsd:attribute ref="xmi:id"/>
    <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
    <xsd:attribute name="a" type="xsd:string" use="optional"/>
  </xsd:complexType>
</xsd:element>
```

An element is also declared to be of XML type string if the class contains a Tag org.omg.xmi.schemaType with value "string."

For multi-valued attributes, no XML attributes are declared; each value is encoded as an XML element.

When "a" is an attribute with enumerated values, the type used for the declaration of the XML element and XML attribute corresponding to the metamodel attribute is as follows:

```
<xsd:simpleType base="enumName" >
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="v1"/>
    <xsd:enumeration value="v2"/>
  </xsd:restriction>
</xsd:simpleType>
```

where **enumName** is the name of the enumeration type, and **v1** and **v2** are the enumeration literals.

If an attribute has enumerated values, an XML element and an XML attribute is put in the complexType for the class "c;" their declaration is as follows:

```
  <xsd:element name="a" type="enumName"/>

  <xsd:attribute name="a" type="enumName" use="optional"/>
```

If an attribute is a multi-valued enumeration attribute, the declaration of the XML attribute is omitted.

In some MOF models, enumerations have a prefix substring that should be removed before placing the enumeration literals in the schema. The Tag "org.omg.xmi.xmiName" indicates the name for the enumeration literal that should be used in XMI documents and schemas.

Default values for property and enumeration attributes may be specified in schemas using the Tag "org.omg.xmi.defaultValue" attached to the attribute. The default value should be the XML string representation to be placed in the schema. Default values for attributes should be specified in schemas with care since XML allows the processor reading the document the option of not processing a schema as an optional optimization. When tools skip processing the schema, they do not obtain the default value of XML attributes. Instead, they would have to know the default value from understanding the metamodel. The form for specifying defaults, where "d" is the default, is:

For string attributes, the corresponding XML attribute declaration in the XML element corresponding to the class is:

```
<xsd:attribute name="a" type="xsd:string" default="d"/>
```

For enumeration attributes, the corresponding XML attribute declaration in the XML element corresponding to the class is:

```
<xsd:attribute name="a" type="enumTypeName" default="d"/>
```

NOTE: When reading documents with XML elements specifying model attribute values, be sure to use the value in the XML element rather than the default value from the unused XML attribute.

4.8.5 Reference Specification

Each reference is represented in an XML element and/or an XML attribute. The XML element declaration for a reference named "r" for a metamodel class "c" of type "classType" is:

```
<xsd:element name="r" type="xmi:Any"/>
```

This element is declared in the content of the complexType for the class that owns the reference. This declaration enables any object to be serialized, enhancing the extensibility of models. A user can override this declaration using the useSchemaExtension tag or the contentType tag.

The attribute declaration for the reference, which also is included in the complexType declaration for the class that owns the reference, is as follows:

```
<xsd:attribute name="r" type="xsd:IDREFS" use="optional"/>
```

4.8.6 Containment Specification

Each association end that represents containment is represented by an XML element, but not by an XML attribute. The form of the XML element is identical to that for association roles.

4.8.7 Inheritance Specification

XML schemas have a mechanism for extending types, but it does not support extending from more than one type, and using that mechanism imposes an order on the content models of the types that are derived from other types. Since XMI attempts to minimize order dependencies, XMI by default does not use schema extension to represent inheritance. In its place, XMI specifies that inheritance will be copy-down inheritance. For attributes and compositions, copy-down inheritance is required. For associations (AssociationEnds with references), the actual class referenced is used, and subclasses may be used on the other end of the reference.

Multiple inheritance is treated in such a way that the attributes, associations, and compositions of classes that occur more than once in the inheritance hierarchy are only included once in their subclasses. For associations (AssociationEnds with references), the actual class referenced is used, and subclasses may be used on the other end of the reference.

4.8.8 Derived Information

Derived information is orthogonal to serialization. The serialization tag is provided to optionally suppress serialized data. This capability provides more control to the end users, allowing them to customize exactly which information is present in their files.

4.9 Transmitting Incomplete Metadata

In XMI 2.0 a schema generator can decide whether to support the exchange of model fragments.

4.9.1 Interchange of Model Fragments

In practice, most information is related. The ability to transfer a subset of known information is essential for practical information interchange. In addition, as information models are developed, they will frequently need to be interchanged before they are complete.

The following guidelines apply for interchanging incomplete models via XMI:

- Information may be missing from a model. The transmission format should not require the addition or invention of new information.
- Model fragments may be disjoint sets. Each set may be transmitted in the same XMI file or in different XMI files.
- "Incomplete" indicates a quantity of information less than or equal to "complete." Additional information beyond that which the metamodel prescribes may be transmitted only via the extension mechanism.
- Semantic verification is performed on the metadata that is actually present as if it was included in complete metadata.

4.9.2 XMI Encoding

The interchange of model fragments is accomplished by lowering the lower bound of multiplicities whose lower bound is greater than 0.

4.9.3 Example

The following is an example of an incomplete UML model:

```
<UML:Model name="model1" xmi:id="id1">
    <ownedElement xmi:type="UML:Class" name="class1" xmi:id="id2">
        <feature xmi:type="UML:Attribute" name="attribute1"
                    type="type1"/>
    </ownedElement>
     <ownedElement xmi:type="UML:Datatype" name="Integer" xmi:id="type1"/>
</UML:Model>
```

4.10 Linking

The goal is to provide a mechanism for specifying references within and across documents. Although based on the XLinks standard, it is downwards compatible and does not require XLinks as a prerequisite.

4.10.1 Design Principles

- Links are based on XLinks to navigate to the document (which may be the current document) and XPointers to navigate to the element within the document.
- Link definitions are encapsulated in the attribute group LinkAttribs defined in 4.6.2, "Linking Attributes," on page 13.
- Elements act as a union, where they are either a definition or a proxy. Proxies use the LinkAttribs attribute group to define the link, and contain no nested elements.
- LinkAttribs supports external links through the XLink attributes, and internal links through the xmi:idref and xmi:id attributes.

- Links are always to elements of the same type or subclasses of that type. Restricting proxies to reference the same element type reduces complexity, enhances reliability and type safety, and promotes caching. In XMI 2.0, subclasses are also allowed, to permit more flexibility in combining models and metamodels.
- When acting as a proxy, XML attributes may be defined, but not contents. The XML attributes act as a cache which gives an indication if the link should be followed.
- Proxies may be chained.
- When following the link from a proxy, the definition of the proxy is replaced by the referenced element.
- It is efficient practice for maximizing caching and encapsulation to use local proxies of the same element within a document to link to a single proxy that holds an external reference.
- Association role elements typically contain proxies that link to the definitions of the classes that participate in the association.

4.10.2 Linking

For XMI, the most common linking requirements are:

- Linking to an XML element in the same document using the element's id.
- Linking to an XML element in a different document using the element's id.
- Linking to an XML element using the element's uuid, in the same or a different document.

The following sub clauses describe how XMI supports these requirements.

Linking within a Document

The **idref** attribute may be used to specify the XML ID of an XML element within the current XML document. Every construct that can be referred to has a local XML ID, a string that is locally unique within a single XML file.

Linking across Documents

Supporting links across documents is optional.

1. Using the XMI href attribute to locate an XMI id.

This is the simplest form of cross document linking. With help from the XMI idName tag, it can be backward compatible with XMI 1.2 and later.

Here, the XMI **href** attribute is used to locate an XML element in another XML document by its XMI id. The value of **href** must be a URI reference, as defined by IETF RFC 2396: Uniform Resource Identifiers. The URI reference must be of the form URI#id_value, where URI locates the XML file containing the XML element to link to, and id_value is the value of the XML element's XMI id attribute.

As an example:

<mgr xmi:id="mgr_1" href="Co.xml#emp_2"/>

locates XML element **<Employee xmi:id="emp_2" …** /> in file Co.xml.

2. Using an XLink simple link and XPointer bare name to locate an XMI id.

This is a little more complicated than using the XMI **href** attribute, and does not provide any more function. It does have the advantage that standard XLink and XPointer software can follow the link.

Here, an xlink:href attribute is used, where XLink is the prefix for the XLink namespace. The XLink prefix must be declared in the document that contains the Xlink:href attribute, for example:

```
<xmi:XMI version="2.0" xmlns:xlink="http://www.w3.org/1999/XLink"
        xmlns:xmi=" http://schema.omg.org/spec/XMI/2.0">
```

The value of xlink:href must again be a URI reference of the form URI#id_value. In this case, id_value is technically an XPointer bare name, but it looks just like the id_value for the XMI href attribute.

The XML element with the xlink:href must also have an xlink:type="simple" attribute, to identify it as a simple link.

As an example:

```
<mgr xmi:id="mgr_1" xlink:href="Co.xml#emp_2" xlink:type="simple"/>
```

locates XML element <**Employee xmi:id="emp_2" …** /> in file Co.xml.

3. Using an XLink simple link and full XPointer to locate an XMI uuid.

An XLink simple link and a form of full XPointer can be used to locate an XML element in an XML document by its XMI uuid. Again:

- An xlink:href attribute is used, where XLink is the prefix for the XLink namespace. The xlink prefix must be declared in the document containing the xlink:href attribute.
- The value of xlink:href must be a URI reference.

However this time, the URI reference has a more complicated form:

URI#xpointer((//*[@xmi:uuid='value'])[1])

The xpointer expression is a series of instructions for finding the first element in the target file whose xmi:uuid has that value.

As an example:

```
<mgr xmi:id="mgr_1"
        xlink:href="Co.xml#xpointer((//*[@xmi:uuid='emp_2'])[1])"
        xlink:type="simple"/>
```

locates XML element <**Employee xmi:uuid="emp_2"…**/> in file Co.xml, as long as it's the first element with that uuid in the file.

Since a URI can identify the same file that contains the href, this also supports locating XML elements by XMI uuid in the same document.

4. Using full XLink and XPointer to locate almost anything.

XLink and XPointer provide rich and complex capabilities for locating XML elements, far beyond what XMI requires. Consequently it is not expected that XMI implementations supporting linking across documents provide this level of support. The W3C XLink and XPointer specifications define what's possible and how it works.

4.10.3 Example from UML

There is an association between ModelElements and Constraints in UML. Operations are a subclass of ModelElements. This example shows an association between Operations and four Constraints with roles constraint and constrainedElement. Each of the methods of linking is shown. The Constraints are shown in both definition and proxy form.

Document 1

```
<UML:Operation xmi:id="idO1" xmi:label="op1" xmi:uuid="DCE:1234">
    <constraint xmi:id="idC1" xmi:label="co1" xmi:uuid="DCE:abcd">
        <body>First Constraint definition</body>
        <constrainedElement xmi:idref="idO1"/>
    </constraint>
    <constraint xmi:idref="idC2" />
    <constraint xmi:idref="idC3" />
    <constraint href="doc2.xml#idC4" />
</UML:Operation>
<UML:Constraint xmi:id="idC2" xmi:label="co2" xmi:uuid="DCE:efgh">
    <body>Second Constraint definition</body>
    <constrainedElement xmi:idref="idO1" />
</UML:Constraint>
<UML:Constraint xmi:id="idC3" xmi:label="co3" xmi:uuid="DCE:ijkl">
    <body>Third Constraint definition</body>
    <constrainedElement
        href="#xpointer(descendent(1,Operation,xmi:label,op1))"/>
</UML:Constraint>
```

Document 2

```
<UML:Constraint xmi:id="idC4" xmi:label="co4" xmi:uuid="DCE:mnop">
    <body>Fourth Constraint definition</body>
    <constrainedElement href="doc1.xml#idO1"/>
</UML:Constraint>
```

The first constraint is a definition. The constrainedElement role contains an Operation proxy that has a local reference to the initial Operation definition using **xmi:idref**. The second constraint is a proxy referencing a constraint definition using the **xmi:idref** of “idC2.” The third constraint is a proxy reference to the definition using **xmi:idref** to the constraint “idC3.” The fourth constraint is an XPointer reference proxy to the definition of the constraint using the **href** to the file doc2.xml with id “idC4.”

Following the definition of the operation and its 3 constraint proxies are the definitions of two of the constraints. The second document contains the third constraint definition.

The use and placement of references is freely determined by the document creator. It is likely that most documents will make internal and external references for a number of reasons: to minimize the amount of duplicate declarations, to compartmentalize the size of the document streams, or to refer to useful information outside the scope of transmission. For example, the **href** of an XLink could contain a query to a repository that will recall additional related information. Or there may be a set of XMI documents created, one file per package to be transferred, where there are relationships between the packages.

4.11 Tailoring Schema Production

This sub clause describes how to tailor schema production by specifying particular MOF tags as part of a MOF metamodel. It also explains the impact the tailored schemas have on document production.

Note that the MOF definition of the association between ModelElement and Tag is not a composition and does not have a reference as part of ModelElement. This allows Tags to be contained in separate Packages and ‘remotely’ reference the tagged elements. For XMI purposes this means that the following tags can be incrementally added to an existing metamodel without needing to be embedded in it - and thus changing it. Typically, the Tags could be in a separate Package and a ‘super’ package could cluster this Tags package and the metamodel package to drive the Schema generation. This conveniently allows different Tag sets to be used with the same metamodel (there would be a separate ‘super’ package for each). And the ‘super’ package extent allows runtime metamodel access to the Tags package for introspection of the tags that were used for the generation.

4.11.1 XMI Tag Values

Table 4.1 specifies the XMI tags that allow you to tailor the schemas that are produced and the documents that are produced using XMI. Each of the names has a prefix of “org.omg.xmi.” but the prefix is not included in the names to make the table easier to read.

Table 4.1- XMI Tag Values Summary

Tag Name	Value Type	Default value	Description
Naming tags			
xmiName	string	nil	Provides an alternate name from the MOF name for writing to XMI. Useful in cases where the MOF name has characters that conflict with XML. This value is used rather than the MOF name.
idName	string	xmi:id	The value is the name of the id attribute.
nsURI	string	nil	The namespace URI of the MOF package.
nsPrefix	string	nil	The namespace prefix of the MOF package; this is used in schemas. (Any legal XML prefix may be used in documents.)
XML Syntax tags			
serialize	boolean	true	If false, suppresses serialization of the MOF construct. Typically applied to derived features.
attribute	boolean	false	If true, serializes the MOF construct as an XML attribute.
element	boolean	false	If true, serializes the MOF construct as an XML element.
remoteOnly	boolean	false	If set on one end of a bidirectional relationship, only serializes that end if it is remote.

Table 4.1- XMI Tag Values Summary

Tag Name	Value Type	Default value	Description
href	boolean	false	If true, use the href attribute rather than the idref attribute for links within a document.
Ordering			
superClassFirst	boolean	false	If true, serialize the super class content first.
ordered	boolean	false	If true, serialize object content in the order it is defined in a MOF metamodel.
Content			
includeNils	boolean	false	If false, do not serialize nil values.
XML Schema Production			
enforceMaximumMultiplicity	boolean	false	If true, enforce maximum multiplicities; otherwise, they are "unbounded."
enforceMinimumMultiplicity	boolean	false	If true, enforce minimum multiplicities; otherwise, they are "0."
useSchemaExtensions	boolean	false	If true, use schema extensions to represent inheritance in the MOF metamodel.
schemaType	string	nil	The name of a datatype defined in the XML Schema Datatype specification.
contentType	string	empty	Defines the schema content type. Other valid values are: complex, any, mixed, complex, and simple.
processContents	string	strict	If the contentType is any, this tag is used to specify the value of the processContents attribute of the any element. Other valid values are: lax, skip.
form	string	nil	Specifies the value of the form attribute for attributes. Other valid values are qualified and unqualified.
defaultValue	string	nil	Specifies the default value for attributes.
fixedValue	string	nil	Specifies the fixed value for attributes.

4.11.2 Tag Value Constraints

There are constraints on the values of the XMI tags in addition to the ones specified in the above table. Here is a list of them:

- If includeNils is true, and the value of an attribute is nil, the value must be represented by an XML element regardless of the value of the attribute tag. Note that MOF references cannot be set to nil.
- If enforceMinimumMultiplicity or enforceMaximumMultiplicity is true, the ordered tag must be true as well (to validate multiplicities, schemas require element content to be serialized in a particular order). The multiplicity tags require the use of serializing in elements.

- If useSchemaExtensions is true, the MOF metamodel must not have multiple inheritance.
- If useSchemaExtensions is true, superClassFirst must be true also.
- If href is true, element must be true as well for every reference that is serialized.
- The attribute tag may not be specified on containment references, multi-valued attributes, attributes without simple data types, or features with the following tags as true: element, includeNils, enforceMinimumMultiplicity, enforceMaximumMultiplicity, and href.

4.11.3 Scope

With the exception of xmiName, serialize, and remoteOnly, all of these tags apply to all constructs within the scope of the construct they are assigned to. If they are specified for a MOF package, they apply to constructs within the scope of the MOF package. If they are specified for a MOF class, they apply to the MOF class and the features of the class. For example, if you set the element tag to true for a MOF class, you should serialize the values of all features of the class using XML elements rather than XML attributes.

The xmiName, serialize, contentType, schemaType, and remoteOnly tags apply only to the constructs for which they are specified. For example, setting the xmiName of a MOF class to “c” means that the name “c” should be used in XMI schemas and documents for that class; it does not constrain the names of the features of the class.

4.11.4 XML element vs XML attribute

You may choose features (MOF attribute or reference) to appear as XML attributes, XML elements, or both based on the model and tags in the model. The following is a list of the conditions for mapping a feature to an XML construct.

XML attribute only

- The feature has an attribute tag set to true.

XML element only

- The feature is a containment reference, or
- the feature has an element tag set to true, or
- the feature has an href tag set to true, or
- the feature is a multi-valued attribute, or
- the feature is an attribute whose type is not a simple data type.

Both XML attribute and element

- The default.

4.11.5 UML profile for XML and XMI

The tags defined above define a UML profile for XML and XMI. The tags placed on a UML element are transferred directly to the corresponding MOF element when converting UML to MOF. In addition, a UML element with a stereotype of one of the above non-prefixed tag names are transferred to a MOF tag of the same name and value true. A UML profile for MOF supplements this profile by providing exact mappings from UML models to MOF models.

An example of the UML profile for XML and XMI would be placing the <<element>> stereotype on a UML attribute that should always be written as an XML element. The corresponding MOF tag would have value true.

4.11.6 Effects on Document Production

The values of the XMI tags affect how documents are serialized. In general, the more validation a schema performs, the more restrictions there are on the XMI documents that validate using the schemas. There are two reasons for this. First, schemas cannot validate multiplicities without imposing an order on element content. Second, if the schema extension mechanism is used, superclass elements must be serialized in element content before subclass elements.

Here are some examples of how the XMI tags affect document production. Assume that there is a MOF metamodel with class “Super” and class “Sub.” Sub inherits from Super. Super has attribute a of type string, and Sub has attribute b of type string. If the namespace URI is “URI,” and the prefix is “p,” here is the default schema produced from the MOF metamodel:

```
<xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="URI"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:p="URI">

<xsd:import
    namespace="http://www.omg.org/XMI"
    schemaLocation="xmi20.xsd"/>

<xsd:complexType name="Super">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="a" type="xsd:string"/>
    <xsd:element ref="xmi:Extension"/>
  </xsd:choice>
  <xsd:attribute ref="xmi:id"/>
  <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
  <xsd:attribute name="a" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:element name="Super" type="p:Super"/>

<xsd:complexType name="Sub">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="a" type="xsd:string"/>
    <xsd:element name="b" type="xsd:string"/>
    <xsd:element ref="xmi:Extension"/>
  </xsd:choice>
  <xsd:attribute ref="xmi:id"/>
  <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
  <xsd:attribute name="a" type="xsd:string" use="optional"/>
  <xsd:attribute name="b" type="xsd:string" use="optional"/>
</xsd:complexType>
```

```
<xsd:element name="Sub" type="p:Sub"/>

</xsd:schema>
```

Note that the content model for **Sub** allows attribute **a** or attribute **b** to be serialized first if they are serialized as elements. For example, if **p** is the namespace prefix for a namespace whose uri is “URI” in an XML document, the following instance of **Sub** validates against the default schema:

```
<p:Sub>
  <b>Value1</b>
  <a>Value2</a>
</p:Sub>
```

The following is also legal:

```
<p:Sub>
  <a>Value2</a>
  <b>Value1</b>
</p:Sub>
```

If useSchemaExtensions is true, the declaration of the Sub complexType uses the XML schema extension mechanism, as follows:

```
<xsd:complexType name="Sub">
  <xsd:complexContent>
    <xsd:extension base="p:Super">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="b" type="xsd:string"/>
      </xsd:choice>
      <xsd:attribute name="b" type="xsd:string" use="optional"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

This declaration of the **Sub** type imposes an ordering on the content of **Sub** instances. With this declaration, attribute **a** must be serialized before attribute **b**, so the first instance of **Sub** above does not validate with this schema, but the second does validate. Also, any xmi:extension elements must be serialized in **Sub** instances before elements corresponding to attribute **b**.

4.11.7 Summary of XMI Tag Scope and Affect

Table 4.2 contains the following information:

- Affect: the second column identifies the MOF constructs that are affected by a given XMI tag.
- Scope: columns 3 through 5 identify the scope of each tag. If the scope is Package Scope, a tag set on the package applies to all the affected constructs within the package. If the scope is Class Scope, a tag set on the class applies to all affected constructs within the class. If the scope is Construct Scope, the tag affects only the specific construct it’s set on.

By setting a tag on a package or class, you avoid setting the same tags repeatedly for classes in the package, and for attributes and association ends belonging to the class. For example, the element tag applies to attributes and association ends. If the element tag is set to true for a class, the class itself is not affected, but each attribute and association end belonging to the class is treated as if the element tag were set to true for all of them.

Table 4.2- XMI Tags, the MOF Constructs They Affect, and Their Scope

XMI Tag	MOF Constructs Affected	Package Scope	Class Scope	Construct Scope
xmiName	Class, Attribute, AssociationEnd			X
serialize	Attribute, AssociationEnd	X	X	X
element	Attribute, AssociationEnd	X	X	X
attribute	Attribute, AssociationEnd	X	X	X
enforceMaximumMultiplicity	Attribute, AssociationEnd	X	X	X
enforceMinimumMultiplicity	Attribute, AssociationEnd	X	X	X
form	Attribute, AssociationEnd	X	X	X
remoteOnly	AssociationEnd	X	X	X
href	AssociationEnd	X	X	X
includeNils	Attribute	X	X	X
schemaType	Attribute			X
defaultValue	Attribute			X
fixedValue	Attribute			X
nsURI	Package	X		X
nsPrefix	Package	X		X
idName	Class	X	X	X
useSchemaExtensions	Class	X	X	X
contentType	Class	X	X	X
superClassFirst	Class	X	X	X

4.12 Transmitting Metadata Differences

The goal is to provide a mechanism for specifying the differences between documents so that an entire document does not need to be transmitted each time. This design does not specify an algorithm for computing the differences, just a form for transmitting them.

Up to now we have seen how to transmit an incomplete or full model. This way of working may not be adequate for all environments. More precisely, we could mention environments where there are many model changes that must be transmitted very quickly to other users. For these environments the full model transmission can be very resource consuming (time, network traffic, ...) making it very difficult or even not viable for finding solutions for cooperative work.

The most viable way to solve this problem is to transmit only the model changes that occur. In this way different instances of a model can be maintained and synchronized more easily and economically. Concurrent work of a group of users becomes possible with a simple mechanism to synchronize models. Transmitting less information allows synchronizing models more efficiently.

4.12.1 Definitions

The idea is to transmit only the changes made to the model (differences between new and old model) together with the necessary information to be able to apply the changes to the old model.

A. New - Old = Difference

Model differencing is the comparison of two models and identifying the differences between them in a reversible fashion. The difference is expressed in terms of changes made to the old document to arrive at the new document.

B. New = Old + Difference

Model merging is the ability to combine difference information plus a common reference model to construct the appropriate new model.

4.12.2 Differences

Differences **must** be applied in the order defined. A later difference may refer to information added by a previous difference by linking to its contents. Model integrity requires that all the differences transmitted are applied. The following are the types of differences recognized, the information transmitted, and the changes they represent:

- Delete (reference to deleted element): The delete operation refers to a particular element of the old model and specifies a deep removal of the referenced element and all of its contents.
- Add (reference to containing element, new element, optional position): The add operation refers to a particular element of the old model and specifies a deep addition. The element and its contents are added. The contents of the new element are added at the optional position specified, the default being as the last element of the contents. The optional position form is based on XPointer's position form. 1 means the first position, -1 means the last position, and higher numbers count across the contents in the specified direction.
- Replace (reference to replaced element, replacement element, optional position): This operation deletes the old element but not its contents. The new element and its contents are added at the position of the old element. The original contents of the old element are then added to the contents of the new element at the optional position specified, the default being at the end.

4.12.3 XMI Encoding

The following are the elements used to encode the differences:

delete

The delete element's link attributes contain a link to the element to be deleted.

add

The contents of add is the element to be added. The link attributes contain a link to the element to be deleted and an optional position element. The numbering corresponds to XPointer numbering, where 1 is the first and -1 is the last element.

replace

The contents of replace is the element to replace the old element with. The attributes contain a link to the element to be replaced and an optional position element for the replacing element's contents. The numbering corresponds to XPointer numbering, where 1 is the first and -1 is the last element.

4.12.4 Example

This example will delete a class and its attributes, add a second class, and rename a package.

The original document:

```
<xmi:XMI version="2.0" xmlns:UML="org.omg/UML"
                    xmlns:xmi="http://www.omg.org/XMI">
    <UML:Package xmi:id="ppp" xmi:label="p1">
        <ownedElement xmi:type="UML:Class" xmi:id="ccc" xmi:label="c1">
            <feature xmi:type="UML:Attribute" xmi:label="a1"/>
            <feature xmi:type="UML:Attribute" xmi:label="a2"/>
        </ownedElement>
    </UML:Package>
</xmi:XMI>
```

The differences document:

```
<xmi:XMI version="2.0" xmlns:UML="org.omg/UML"
                                xmlns:xmi="http://www.omg.org/XMI">
    <difference xmi:type="xmi:Delete">
      <target href="original.xml#ccc"/>
     </difference/>
    <difference xmi:type="xmi:Add" addition="Class_1">
      <target href="original.xml#ppp"/>
     </difference>
    <UML:Class xmi:id="Class_1" xmi:label="c2"/>
    <difference xmi:type="xmi:Replace" replacement="ppp">
      <target href="original.xml#ppp"/>
     </difference>
    <UML:Package xmi:id="ppp" xmi:label="p2"/>
</xmi:XMI>
```

Here's how the 3 differences change the document as they're applied.

The delete:

```
<xmi:XMI version="2.0" xmlns:UML="org.omg/UML"
                                xmlns:xmi="http://www.omg.org/XMI">
  <UML:Package xmi:id="ppp" xmi:label="p1"/>
</xmi:XMI>
```

Next, the add:

```
<xmi:XMI version="2.0" xmlns:UML="org.omg/UML"
                      xmlns:xmi="http://www.omg.org/XMI">
  <UML:Package xmi:id="ppp" xmi:label="p1">
```

```
    <ownedElement xmi:type="UML:Class" xmi:label="c2"/>
  </UML:Package>
</xmi:XMI>
```

Finally, the replace:

```
<xmi:XMI version="2.0" xmlns:UML="org.omg/UML"
                       xmlns:xmi="http://www.omg.org/XMI">
  <UML:Package xmi:id="ppp" xmi:label="p2">
    <ownedElement xmi:type="UML:Class" xmi:label="c2"/>
  </UML:Package>
</xmi:XMI>
```

4.13 Document Exchange with Multiple Tools

This sub clause contains a recommendation for an optional methodology that can be used when multiple tools interchange documents. In this methodology, the **xmi:uuid** and extensions are used together to preserve tool-specific information. In particular, tools may have particular requirements on their IDs that make ID interchange difficult. Extensions are used to hold tool-specific information, including tool-specific IDs.

The basic policy is that the XML ID is assigned by the tool that initially creates a construct. The **UUID** will most likely be the same as the ID the tool would choose for its own use. Any other modifiers of the document must preserve the original **UUID**, but may add their own as part of their extensions.

4.13.1 Definitions

General:

- MC - Model construct. An XML element that contains an xmi.uuid attribute.
- Extension - Extensions use the extension element. Extensions to MCs may be nested in MCs, linked to the extensions area(s) of the document, or linked outside the document. Each extension contains a tool-specific identifier in the extender attribute. Extensions are considered private to a particular tool. An MC may have zero or more extensions. Extensions may be nested.

IDs:

- **xmi:uuid** - The universally unique ID of an MC, expressed as the **xmi:uuid** attribute. Example: <Class **xmi:uuid**=”ABCDEFGH”>
- **extenderID** - The tool-specific ID of an MC. The extenderID is stored in an extension of the MC when it differs from the **xmi:uuid**.

Tool ID policies:

Every tool is either Open or Closed.

- Open tool - A tool that will accept any **xmi:uuid** as its own. Open tools do not need to add extensions to contain a tool-specific id.
- Closed tool - A tool that will not accept an **xmi:uuid** created by another tool. Closed tools store their ids in the **extenderID** attribute of an XMI.extension. The **extender** attribute of the XMI.extension is set to the name of the closed tool.

4.13.2 Procedures

Document Creation:

The Creating Tool writes a new XMI document. Each MC is assigned an **xmi:uuid**. If the **xmi:uuid** differs from the **extenderID**, an **extension** for that tool is added containing the extenderID.

Document Import:

The importing tool reads an existing XMI document. Extensions from other tools may be stored internally but not interpreted in the event a Modification will occur at a later time. One of the following cases occurs:

- If the importing tool is an Open tool, the **xmi:uuid**s are accepted internally and no conversion is needed.
- If the importing tool is a Closed tool, the tool looks for a contained extension (identified by extender) with an extenderID. If one does not exist, the importing tool creates its own internal id.

Document Modification:

- The modifying tool writes the MCs and any extensions preserved from import.
- For new MCs, the MC is assigned an **xmi:uuid**.
- Closed tools add an **extension** including their internal id in the **extenderID**.

4.13.3 Example

This sub clause describes a scenario in which Tool1 creates an XMI document that is imported by Tool2, then exported to Tool1, and then a third tool imports the document. All the tools are closed tools.

1. A model is created in Tool1 with one class and written in XMI.

```
<UML:Class xmi:label="c1" xmi:uuid="abcdefgh"/>
```

2. The class is imported into Tool2. Tool2 assigns extenderID “JKLMNOPQRST.” A second class is added with name “c2” and extenderID “X012345678.”
3. The model is merged back to XMI:

```
<UML:Class xmi:label="c1" xmi:uuid="abcdefgh">
                <xmi:Extension extender="Tool2" extenderID="JKLMNOPQRST"/>
    </UML:Class>
    <UML:Class xmi:label="c2" xmi:uuid="X012345678"/>
```

4. The model is imported into Tool1. Tool1 assigns extenderID “ijklmnop” to “c2” and a new class “c3” is created with extenderID “qrstuvwxyz.”
5. The model is merged back to XMI:

```
<UML:Class xmi:label="c1" xmi:uuid="abcdefgh">
                <xmi:Extension extender="Tool2" extenderID="JKLMNOPQRST"/>
        </UML:Class>
        <UML:Class xmi:label="c2" xmi:uuid="X012345678">
              <xmi:Extension extender="Tool1" extenderID="ijklmnop"/>
        </UML:Class>
        <UML:Class xmi:label="c3" xmi:uuid="qrstuvwxyz"/>
```

6. A third closed tool, Tool3, adds its ids:

```
<UML:Class xmi:label="c1" xmi:uuid="abcdefgh">
            <xmi:Extension extender="Tool2" extenderID="JKLMNOPQRST"/>
            <xmi:Extension extender="Tool3" extenderID="s1234"/>
        </UML:Class>
        <UML:Class xmi:label="c2" xmi:uuid="X012345678">
             <xmi:Extension extender="Tool1" extenderID="ijklmnop"/>
             <xmi:Extension extender="Tool3" extenderID="s5678"/>
        </UML:Class>
        <UML:Class xmi:label="c3" xmi:uuid="qrstuvwxyz">
              <xmi:Extension extender="Tool3" extenderID="s90ab"/>
        </UML:Class>
```

7. An open tool imports and modifies the file. There are no changes because the **xmi:uuids** are used by the tool.

4.14 General Datatype Mechanism

The ability to support general data types in XMI has significant benefits. The applicability of XMI is significantly expanded since domain metamodels are likely to have a set of domain-specific data types. This general solution allows the user to provide a domain datatype metamodel with a defined mapping to the XML data types.

Data types are defined in the model and the XML serialization of the datatypes is described in terms of the XML schema datatypes.

MOF complex data types are treated as MOF classes with each field treated as a MOF attribute with a primitive type mapped to XML schema.

The Tag org.omg.xmi.schemaType indicates that this class is a datatype with XML schema mapping. The value of the tag indicates the schema type. For example, http://www.w3.org/2001/XMLSchema#int is the int datatype.

(Blank page)

5 XML Schema Production

5.1 Purpose

This sub clause describes the rules for creating a schema from a MOF-based metamodel. The conformance rules are stated in Annex A.

Notation for EBNF

The rule sets are stated in EBNF notation. Each rule is numbered for reference. Rules are written as rule number, rule name, for example 1a. SchemaStart. Text within quotation marks are literal values, for example "<xsd:element." Text enclosed in double slashes represents a placeholder to be filled in with the appropriate external value, for example //Name of Attribute//. Literals should be enclosed in single or double quotation marks when used as the values for XML attributes in XML documents. The suffix "*" is used to indicate repetition of an item 0 or more times. The suffix "?" is used to indicate repetition of an item 0 or 1 times. The suffix "+" is used to indicate repetition of an item 1 or more times. The vertical bar "|" indicates a choice between two items. Parentheses "()" are used for grouping items together.

EBNF ignores white space; hence these rules do not specify white space treatment. However, since white space in XML is significant, the actual schema generation process must insert white space at the appropriate points.

5.2 XMI Version 2 Schemas

5.2.1 EBNF

The EBNF for XMI Version 2 schemas is listed below with rule descriptions between sub clauses:

```
1.  Schema              ::= 1a:SchemaStart
                            1d:ImportsAndIncludes?
                            1e:FixedDeclarations
                            2:PackageSchema+
                            1f:SchemaEnd
1a. SchemaStart         ::= "<xsd:schema
                                 xmlns:xsd='http://www.w3.org/2001/XMLSchema'
                                 xmlns:xmi='http://www.omg.org/XMI'"
                                1b:NamespaceDecl*
                                1c:TargetNamespace?
                               ">"
1b. NamespaceDecl       ::= "xmlns:" //Namespace name// "="
                               "'" //Namespace URI// "'"
1c. TargetNamespace     ::= "targetNamespace='" //Namespace URI// "'"
1d. ImportsAndIncludes::= // Imports and includes //
1e. FixedDeclarations  ::= "<xsd:import
                                  namespace='http://www.omg.org/XMI'/>"
1f. SchemaEnd           ::= "</xsd:schema>"
1g. XMIFixedAttribs     ::= ( "<xsd:attribute ref='xmi:id'"
                                          "use='optional'>" |
                                "<attribute name='" // Id attrib name // "'"
                                            "type='xsd:ID' use='optional'")
                               "<xsd:attributeGroup ref='xmi:ObjectAttribs'/>"
1h. Namespace           ::= ( //Name of namespace// ":" )?
```

1.	A schema consists of a schema XML element that contains import and include statements, fixed declarations, plus declarations for the contents of the Packages in the metamodel.
1a.	The schema XML element consists of the schema namespace attribute, namespace attributes for the other namespaces used in the schema, if any, and an optional target namespace attribute. These rules are written as if the namespace name for the schema namespace is “xsd” and the namespace name for the XMI namespace is “xmi,” but you can substitute other names for these namespace names and still conform to this specification.
1b.	Each namespace used in the schema must have a namespace attribute that identifies the namespace name and the namespace URI. If the namespace name is "" , the attribute name should be “xmlns.” The namespace is declared by the nsPrefix and nsURI tags in the metamodel.
1c.	If the schema has a target namespace, the targetNamespace attribute is present.
1d.	If the schema uses declarations from other schemas, the appropriate include or import statements must be present.
1e.	The schema declarations that are in the XMI namespace are listed in 6.3.2, “Overall Content Structure,” on page 51.

1f.	The end of the schema XML element.
1g.	The fixed XMI attributes present on the major elements provide element identity and element linking. If the org.omg.xmi.idName tag has a value, that value is the name of the ID attribute; otherwise, the name is “xmi:id”.
1h.	A namespace is a namespace name followed by a “:”. If no namespace name is given, the rule is a blank.

```
2.  PackageSchema     ::= ( 2:PackageSchema
                            | 3:ClassSchema
                            | 13:EnumSchema)*
                           6:PackageElementDef
```

2.	The schema contribution from a Package consists of the declarations for any contained Packages, Classes, Associations without References, enumerations, and an XML element declaration for the Package itself.

```
3.  ClassSchema       ::= 4:ClassTypeDef
                          5:ClassElementDef
```

3.	The class schema contribution consists of a type declaration based on the attributes and references of the class, and an element declaration for the Class itself.

```
4.  ClassTypeDef   ::= "<xsd:complexType name='" //Name of Class//
                            ("mixed='true'")?
                         "'>"
                           ( "<xsd:complexContent>"
                             "<xsd:extension base='" 4a:ClassTypeName "'")?
                           ("<xsd:choice minOccurs='0'
                                         maxOccurs='unbounded'>" |
                             "<xsd:sequence>")?
                           ( 4b:ClassContents |
                                "<xsd:any minOccurs='0' maxOccurs='unbounded'
                                   processContents='" // ProcessContents Value //
                              "'/>")?
                           ("</xsd:choice>" | "</xsd:sequence>")?
                           4g:ClassAttListItems
                           ( "</xsd:extension>"
                             "</xsd:complexContent>" )?
                           "</xsd:complexType>
4a. ClassTypeName     ::= 1h:Namespace //Name of Class//
4b. ClassContents     ::= 4d:ClassAttributes
                          4e:ClassReferences
                          4f:ClassCompositions
                          4c:Extension
4c. Extension         ::= ("<xsd:element ref='xmi:extension'/>")*
4d. ClassAttributes   ::= ("<xsd:element name='" //Name of Attribute// "'"
                              ("nillable='true'")?
                              ( 4m:MinOccursAttrib )?
                              ( 4n:MaxOccursAttrib )?
                              (("type='" //Name of type// "'/>") |
                               ("type='xmi:Any'/>")) )*
4e. ClassReferences   ::= ( "<xsd:element name='" //Name of Reference// "'"
                              ( 4m:MinOccursAttrib )?
                              ( 4n:MaxOccursAttrib )?
                              (("type='" 4a:ClassTypeName "'/>") |
                              ("type='xmi:Any'/>")) )*
4f. ClassCompositions ::= ( "<xsd:element name='" //Name of Reference// "'"
                               ( 4m:MinOccursAttrib )?
                               ( 4n:MaxOccursAttrib )?
                               (("type='" 4a:ClassTypeName "'/>") |
                                              ("type='xmi:Any'/>")) )*
4g. ClassAttListItems ::= 1g:XMIFixedAttribs 4h:ClassAttribAtts
4h. ClassAttribAtts   ::= ( 4i:ClassAttribRef
                            | 4j:ClassAttribData
                            | 4k:ClassAttribEnum )*
4i. ClassAttribRef    ::= "<xsd:attribute name='" //Name of attribute// "'"
                           ("type='xsd:IDREFS' use='optional'/>" |
                            "type='xsd:IDREF' use='required'/>")
4j. ClassAttribData   ::= "<xsd:attribute name='" //Name of attribute// "'"
```

```
                                   "type='xsd:string' "
                                   ("use='optional'" | "use='required'")
                                     ("default='" 4l:ClassAttribDflt "'")?
                                     ("fixed='" 4p:ClassAttribFixed "'")?
                                     ("form='" // Form value // "'")?
                                  "/>"
4k. ClassAttribEnum   ::= "<xsd:attribute name='" //Name of attribute// "'"
                                        "type='" 8a:EnumTypeName "'"
                                      (("use='default'"
                                         "value='" 4l:ClassAttribDflt "'") |
                                         ("use='optional'" | "use='required'")) "/>"
4l. ClassAttribDflt   ::= //Default value//
4m. MinOccursAttrib   ::= "minOccurs='" // Minimum // "'"
4n. MaxOccursAttrib   ::= "maxOccurs='" // Maximum // "'"
4o. Class AttribFixed ::= //Fixed value//
```

4.	These rules describe the declaration of a Class in the metamodel as an XML complex type with a content model and XML attributes. If either of the tags org.omg.xmi.enforceMaximumMultiplicity or org.omg.xmi.enforceMinimumMultiplicity is true, the contents of the class are put in a sequence; otherwise, they are put in a choice. If the org.omg.xmi.contentType tag is complex, the class content declarations appear as defined by rule 4b; however, if the contentType value is empty (the default), they do not appear, and if the contentType value is any, the "xsd:any" element declaration appears instead of the class content. If the contentType value is mixed, then the mixed attribute is included. If org.omg.xmi.useSchemaExtensions is true, the complex type for the class is derived by extension from the complex type for its superclass.
4a.	This rule is for a reference to the type for the class, which is the name of the Class prefixed by the namespace, if present and not the default namespace.
4b. 4c.	The complex type for the Class contains XML elements for the contained Attributes, References and Compositions of the Class, plus an extension element, regardless of whether they are marked as derived. The org.omg.xmi.serialize tag can be used to control whether these constructs are serialized. If org.omg.xmi.useSchemaExtensions is false or not present, inherited Attributes, References, and Compositions are included; otherwise, only local Attributes, References, and Compositions are included.
4d.	The XML element name for each Attribute of the Class is listed as part of the content model of the Class element. This includes the Attributes defined for the Class itself as well as all of the Attributes inherited from superclasses of the Class. The type is "xsd:string" for simple attributes, the name of an enumeration for enumerated attributes, or part of the value of the org.omg.xmi.schemaType tag, if present. If the org.omg.xmi.includeNils tag is false, then the "nillable" attribute is not included in the declaration. If org.omg.xmi.enforceMinimumMultiplicity is true, the minOccurs attribute is included. If org.omg.xmi.enforceMaximumMultiplicity is true, the maxOccurs attribute is included.

4e.	The XML element name for each Reference of the Class is listed in the content model of the Class. The list includes the References defined for the Class itself, as well as all References inherited from the superclasses of the Class. The type is the class name for the Reference type if org.omg.xmi.useSchemaExtensions is "true" or if the org.omg.xmi.contentType is "complex;" otherwise, the type allows any object to be serialized. If org.omg.xmi.enforceMinimumMultiplicity is true, the minOccurs attribute is included. If org.omg.xmi.enforceMaximumMultiplicity is true, the maxOccurs attribute is included.
4f.	The XML element name for each Reference of the Class that is a composite Reference is listed in the content model of the class. The list includes the References defined for the Class itself, as well as all References inherited from the superclasses of the Class. The type is the class name for the Reference type if org.omg.xmi.useSchemaExtensions is "true" or if the org.omg.xmi.contentType is "complex;" otherwise, the type allows any object to be serialized. If org.omg.xmi.enforceMinimumMultiplicity is true, the minOccurs attribute is included. If org.omg.xmi.enforceMaximumMultiplicity is true, the maxOccurs attribute is included.
4g. 4h.	In addition to the standard identification and linkage attributes, the attribute list of the Class element can contain XML attributes for the Attributes and non-composite References of the Class, when the limited facilities of the XML attribute syntax allow expression of the necessary values. Inherited attributes and references are included unless the org.omg.xmi.useSchemaExtensions tag is true, in which case only local attributes and references are included.
4i.	References can be expressed as XML id reference XML attributes. If the multiplicity of the attribute is exactly one, and org.omg.xmi.enforceMinimumMultiplicity is true, the type is IDREF and the attribute is required.
4j.	Single-valued Attributes of a Class that have a string representation for their data are mapped to XML attributes of type "xsd:string", unless the org.omg.xmi.schemaType tag is present, in which case its value is used for the type. Multi-valued Attributes of a Class cannot be so expressed, since the XML attribute syntax does not allow repetition of values. If the multiplicity of the attribute is exactly one, and org.omg.xmi.enforceMinimumMultiplicity is true, the attribute is required to be present.
4k.	Single-valued Attributes that have enumerated values are mapped to XML attributes whose type is the enumerated type. If the multiplicity of the attribute is exactly one, and org.omg.xmi.enforceMinimumMultiplicity is true, the attribute is required to be present.
4l.	If an Attribute is expressed as an XML attribute, its default value may be expressed in the schema if there is a MOF Tag "org.omg.xmi.defaultValue" attached to the Attribute. The value of this Tag must be expressible as an XML attribute string.
4m.	The value for minimum is the minimum multiplicity.
4n.	The value for maximum is the maximum multiplicity.
4o.	If an Attribute is expressed as an XML attribute, its fixed value may be expressed in the schema if there is a MOF Tag "org.omg.xmi.fixedValue" attached to the Attribute. The value of this Tag must be expressible as an XML attribute string.

```
5.  ClassElementDef ::= "<xsd:element name='" //Name of class// "'"
                              "type=' 4a:ClassTypeName "'/>"
```

5.	This rule declares an XML element for a class in a metamodel.

```
6.  PackageElementDef ::= "<xsd:element name='" //Name of package// "'>"
                             "<xsd:complexType>
                               <xsd:choice minOccurs='0' maxOccurs='unbounded'>"
                             6b:PkgContents
                             "</xsd:choice>"
                            6g:PkgAttListItems
                             "</xsd:complexType>
                              </xsd:element>"
6a. PkgElmtName        ::= 1h:Namespace //Name of package//

6b. PkgContents        ::= 6c:PkgAttributes
                           6d:PkgClasses
                           6e:PkgAssociations
                           6f:PkgPackages
                           4c:Extension
6c. PkgAttributes      ::= ( "<xsd:element name='"
                               //Qualified name of Attribute// "'"
                               "type=' //Name of type// "'/>")*
6d. PkgClasses         ::= ( "<xsd:element ref='" 4a:ClassTypeName "'/>")*
6e. PkgAssociations    ::= ( 7:AssociationDef )*
6f. PkgPackages       ::= ( "<xsd:element ref='" 6a:PkgElmtName "'/>")*
6g. PkgAttListItems   ::= 1g:XMIFixedAttribs 6h:PkgAttribAtts
6h. PkgAttribAtts     ::= 4h:ClassAttribAtts
```

6.	The schema contribution from the Package consists of an XML element definition for the Package, with a content model specifying the contents of the Package.
6a.	This rule is for the use of a package name.
6b.	The Package contents consist of any classifier level Attributes, Associations without References, Classes, nested Packages, and an extension.
6c.	Classifier level Attributes of a Package are also known as static attributes. Such Attributes inherited from Packages from which this Package is derived are also included.
6d.	Each Class in the Package is listed. Classes contained in Packages from which this Package is derived are also included.
6e.	It is possible that the Package contains Associations that have no References (i.e., no Class contains a Reference that refers to an AssociationEnd owned by the Association). Every such Association contained in the Package or Package from which the Package is derived is listed as part of the Package contents in order that its information can be transmitted as part of the XML document.
6f.	Nested Packages are listed. Nested Packages included in Packages from which this Package is derived are also included.
6g. 6h.	The Package XML attributes are the fixed identity and linking XML attributes, as well as the XML attribute declarations corresponding to the classifier-level attributes for the classes in the package.

```
7. AssociationDef     ::= "<xsd:element name='" //Name of association// "'>"
                             "<xsd:complexType>
                              <xsd:choice minOccurs='0' maxOccurs='unbounded'>"
                             7b:AssnContents
                              "</xsd:choice>"
                             7d:AssnAtts
                               "</xsd:complexType>
                                </xsd:element>"
7a. AssnElmtName       ::= 1c:Namespace //Name of association//
7b. AssnContents       ::= 7c:AssnEndDef
                           7c:AssnEndDef
                           4c:Extension
7c. AssnEndDef         ::= "<xsd:element"
                             "name='" //Name of association end// "'>"
                              "<xsd:complexType>"
                                  1g:XMIFixedAttribs
                              "</xsd:complexType>"
                              "</xsd:element>"
7d. AssnAtts           ::= 1g:XMIFixedAttribs
```

7.	The declaration of an unreferenced Association consists of the names of its AssociationEnd XML elements.
7a.	The use of the name of the XML element representing the Association.
7b.	The contents of an Association element are its 2 AssociationEnd elements.
7c.	An AssociationEnd element for an unreferenced Association will always be a single identifier reference using the standard fixed attributes (e.g., idref): there is no possibility of nested elements (it is not a composition) nor of varying multiplicity since only simple links (pairs of references) are being represented here.
7d.	The fixed identity and linking XML attributes are the Association XML attributes.

```
8.  EnumSchema       ::= "<xsd:simpleType name='" 8b:EnumName "'>"
                           "<xsd:restriction base='xsd:string'>"
                              8c:EnumLiterals
                            "</xsd:restriction>"
                             "</xsd:simpleType>"
8a. EnumTypeName     ::= 1h:Namespace 8b:EnumName
8b. EnumName         ::= // Name of enumeration //
8c. EnumLiterals     ::= ("<xsd:enumeration value='" 8d:EnumLiteral "'/>")+
8d. EnumLiteral      ::= // Name of enumeration literal //
```

8.	The enumeration schema contribution consists of a simple type derived from string whose legal values are the enumeration literals.
8a.	The name of the enumeration in XML schema references.
8b. 8c.	Each enumeration literal is put in the value XML attribute of an enumeration XML element.
8d.	The name of the enumeration literal.

5.2.2 Fixed Schema Declarations

There are some elements of the schema that are fixed, constituting a form of "boilerplate" necessary for every XMI 2.0 schema. These elements are described in this sub clause. These declarations are in the namespace "http://www.omg.org/XMI."

Only the schema content of the fixed declarations is given here. For a complete description of the semantics of these declarations, see Clause 7.

The fixed declarations are:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns="http://www.omg.org/XMI"
          targetNamespace="http://www.omg.org/XMI">

<xsd:annotation>
  <xsd:documentation>
    The following attribute and attribute group declarations are included
    in the types for MOF classes, but they are not defined in the XMI
    model.
  </xsd:documentation>
</xsd:annotation>

<xsd:attribute name="id" type="xsd:ID"/>

<xsd:attributeGroup name="IdentityAttribs">
   <xsd:attribute name="label" type="xsd:string" use="optional"
                 form="qualified"/>
   <xsd:attribute name="uuid" type="xsd:string" use="optional"
                 form="qualified"/>
</xsd:attributeGroup>

<xsd:attributeGroup name="LinkAttribs">
   <xsd:attribute name="href" type="xsd:string" use="optional"/>
   <xsd:attribute name="idref" type="xsd:IDREF" use="optional"
                 form="qualified"/>
</xsd:attributeGroup>

<xsd:attributeGroup name="ObjectAttribs">
   <xsd:attributeGroup ref="IdentityAttribs"/>
   <xsd:attributeGroup ref="LinkAttribs"/>
   <xsd:attribute name="version" type="xsd:string" use="optional"
                 fixed="2.0" form="qualified"/>
   <xsd:attribute name="type" type="xsd:QName" use="optional"
                 form="qualified"/>
</xsd:attributeGroup>

<xsd:annotation>
  <xsd:documentation>PACKAGE: XMIPackage</xsd:documentation>
</xsd:annotation>

<xsd:annotation>
  <xsd:documentation>CLASS: XMI</xsd:documentation>
</xsd:annotation>

<xsd:complexType name="XMI">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:any processContents="strict"/>
  </xsd:choice>
  <xsd:attribute ref="id"/>
  <xsd:attributeGroup ref="IdentityAttribs"/>
  <xsd:attributeGroup ref="LinkAttribs"/>
  <xsd:attribute name="type" type="xsd:QName" use="optional"
             form="qualified"/>
  <xsd:attribute name="version" type="xsd:string" use="required"
              fixed="2.0" form="qualified"/>
</xsd:complexType>

<xsd:element name="XMI" type="XMI"/>

<xsd:annotation>
  <xsd:documentation>CLASS: Documentation</xsd:documentation>
</xsd:annotation>

<xsd:complexType name="Documentation">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="contact" type="xsd:string"/>
    <xsd:element name="exporter" type="xsd:string"/>
    <xsd:element name="exporterVersion" type="xsd:string"/>
    <xsd:element name="longDescription" type="xsd:string"/>
    <xsd:element name="shortDescription" type="xsd:string"/>
    <xsd:element name="notice" type="xsd:string"/>
    <xsd:element name="owner" type="xsd:string"/>
    <xsd:element ref="Extension"/>
  </xsd:choice>
  <xsd:attribute ref="id"/>
  <xsd:attributeGroup ref="ObjectAttribs"/>
  <xsd:attribute name="contact" type="xsd:string" use="optional"/>
  <xsd:attribute name="exporter" type="xsd:string" use="optional"/>
  <xsd:attribute name="exporterVersion" type="xsd:string"
              use="optional"/>
  <xsd:attribute name="longDescription" type="xsd:string"
              use="optional"/>
  <xsd:attribute name="shortDescription" type="xsd:string"
              use="optional"/>
  <xsd:attribute name="notice" type="xsd:string" use="optional"/>
  <xsd:attribute name="owner" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:element name="Documentation" type="Documentation"/>

<xsd:annotation>
  <xsd:documentation>CLASS: Extension</xsd:documentation>
</xsd:annotation>

<xsd:complexType name="Extension">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
```

```
    <xsd:any processContents="lax"/>
  </xsd:choice>
  <xsd:attribute ref="id"/>
  <xsd:attributeGroup ref="ObjectAttribs"/>
  <xsd:attribute name="extender" type="xsd:string" use="optional"/>
  <xsd:attribute name="extenderID" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:element name="Extension" type="Extension"/>

<xsd:annotation>
  <xsd:documentation>CLASS: Difference</xsd:documentation>
</xsd:annotation>

<xsd:complexType name="Difference">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="target">
      <xsd:complexType>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:any processContents="skip"/>
        </xsd:choice>
        <xsd:anyAttribute processContents="skip"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="difference" type="Difference"/>
    <xsd:element name="container" type="Difference"/>
    <xsd:element ref="Extension"/>
  </xsd:choice>
  <xsd:attribute ref="id"/>
  <xsd:attributeGroup ref="ObjectAttribs"/>
  <xsd:attribute name="target" type="xsd:IDREFS" use="optional"/>
  <xsd:attribute name="container" type="xsd:IDREFS" use="optional"/>
</xsd:complexType>

<xsd:element name="Difference" type="Difference"/>

<xsd:annotation>
  <xsd:documentation>CLASS: Add</xsd:documentation>
</xsd:annotation>

<xsd:complexType name="Add">
  <xsd:complexContent>
    <xsd:extension base="Difference">
      <xsd:attribute name="position" type="xsd:string" use="optional"/>
      <xsd:attribute name="addition" type="xsd:IDREFS" use="optional"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

```
<xsd:element name="Add" type="Add"/>

<xsd:annotation>
  <xsd:documentation>CLASS: Replace</xsd:documentation>
</xsd:annotation>

<xsd:complexType name="Replace">
  <xsd:complexContent>
    <xsd:extension base="Difference">
      <xsd:attribute name="position" type="xsd:string" use="optional"/>
      <xsd:attribute name="replacement" type="xsd:IDREFS"
                     use="optional"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Replace" type="Replace"/>

<xsd:annotation>
  <xsd:documentation>CLASS: Delete</xsd:documentation>
</xsd:annotation>

<xsd:complexType name="Delete">
  <xsd:complexContent>
    <xsd:extension base="Difference"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Delete" type="Delete"/>
<xsd:complexType name="Any">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
     <xsd:any processContents="skip"/>
  </xsd:choice>
  <xsd:anyAttribute processContents="skip"/>
</xsd:complexType>

<xsd:element name="XMIPackage">
  <xsd:complexType>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="Difference"/>
      <xsd:element ref="Add"/>
      <xsd:element ref="Replace"/>
      <xsd:element ref="Delete"/>
      <xsd:element ref="XMI"/>
      <xsd:element ref="Documentation"/>
      <xsd:element ref="Extension"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

```
</xsd:schema>
```

5.2.3 Schema Production Rules for Non-Primitive Data

MOF 1.4 added a set of non-primitive data types. The schema production rules for these data types are defined using the existing production rules in 5.2.1, “EBNF,” on page 35. They are described in the following sub clauses.

Structure Type

The schema production rules for a structure type with structure fields are the same as for a class with attributes. The production rules for classes are defined starting with 3:ClassSchema. For structure types, use the structure type name instead of class name, and the structure field names instead of attribute names.

Enumeration Type

The schema production rules for enumeration types are defined starting with 8:EnumSchema.

Alias Type

The schema production rules for an alias type are the same as for its base type, but using the alias type name instead of the base type name.

Collection Type

The schema production rules for a collection type are the same as for a class that has one attribute with the same type and multiplicity as the collection type. The production rules for classes are defined starting with 3:ClassSchema. For collection types, use the collection type name instead of class name. Use the collection type’s type name instead of attribute name.

6 XML Document Production

6.1 Purpose

This Clause specifies the XMI Version 2 production of an XML document from a MOF model. XMI Version 2 describes an XML syntax that leverages the new capability of XML schema, resulting in smaller, more powerful documents and enhanced human readability. A set of MOF objects are written to an XML document following the grammar defined here. It is essential for successful model interchange that this specification be complete and unambiguous. It is also essential that all significant aspects of the metadata are included in the XML document and can be recovered from it.

6.2 Introduction

XMI's XML document production process is defined as a set of production rules. When these rules are applied to a model or model fragment, the result is an XML document. The inverse of these rules can be applied to an XML document to reconstruct the model or model fragment. In both cases, the rules are implicitly applied in the context of the specific metamodel for the metadata being interchanged.

The production rules are provided as a specification of the XML document production and consumption processes. They should not be viewed as prescribing any particular algorithm for XML producer or consumer implementations.

6.4, "Additional Examples," on page 60 contains additional examples beyond those in the EBNF.

6.3 EBNF Rules Representation

The XML produced by XMI is represented here in Extended Backus Naur Form (EBNF). The XML specification does not require XML processors to preserve the order of XML attributes within an XML element. Therefore, although this grammar indicates that XML attributes should be serialized in a particular order for each XML element, the XML attributes may be serialized in any order. Also, XML attributes are normalized by XML processors, so whitespace may not be preserved. You may choose to serialize parts of objects as XML elements rather than XML attributes using the org.omg.xmi.element tag, as explained below.

The following sub clauses provide the production rules.

6.3.1 Overall Document Structure

```
1:Document                              ::= 1a:XMI | 2:ContentElements

1a:XMI                                  ::= "<" 1b:XMINamespace "XMI" 1c:StartAttribs ">"
                             ( 2:ContentElements )?
                             ( 5j:Extension )*
                             "</" 1b:XMINamespace "XMI>"
1b:XMINamespace      ::= (//NsName// ":") ?
1c:StartAttribs      ::= 1d:XMIVersion 1e:Namespaces
1d:XMIVersion        ::= 1b:XMINamespace "version='" //XMIVersion// "'"
1e:Namespaces        ::= 1f:XMINamespaceDecl ?
                             ( "xmlns:" 1h:NsName "='" 1i:NsURI "'" )*
1f:XMINamespaceDecl  ::= "xmlns='http://www.omg.org/XMI'" |
                         "xmlns:" //NsName// "='http://www.omg.org/XMI'"
1g:Namespace         ::= ( 1h:NsName> ":" )?
1h:NsName            ::= //Name of namespace//
1i:NsURI             ::= //URI of namespace//
```

1.	The content of an XMI document may be enclosed in an XMI XML element, but it does not need to be. The XML specification requires that there be one root element in an XML document for the document to be well-formed.
1a.	An XMI element has XML attributes that declare namespaces and specify the version of XMI, and the XMI element contains XML elements that make up the header, content, differences, and extensions for the XMI document.
1b.	This rule represents the use of the XMI namespace name, XMINsName, in an XMI document. If NsName is "", this rule produces ""; otherwise, this rule produces NsName followed by ":". For example, if the XMI namespace name is "xmi", then the XML element specified in the XMI production rule has a tag name of "xmi:XMI." If the XMI namespace name is "", then the XML element specified in the XMI production rule has a tag name of "XMI."
1c.	The start attributes include the version attribute and the declaration of namespaces used in the document.
1d.	The version must be “2.0” for XMI documents that conform to this specification.
1e.	The XMI namespace and the namespaces associated with a model must be declared or already be visible to the XMI element in the XML document. Since there is no requirement that the XMI XML element be the root element, these namespaces may be declared in XML elements that contain the XMI element.

1g.	The use of a namespace name, including a “:” separator. If the namespace name is blank, the result is the empty string.
1h.	A particular namespace name. Document producers can choose their own namespace names, as long as doing so results in legal XML documents, or they may choose to use the value of the org.omg.xmi.nsPrefix tag.
1i.	The logical URI of the namespace. Note that namespaces are resolved to logical URIs, as opposed to physical ones, so that there is no expectation that this URI will be resolved and that there will be any information at that location. The URI is obtained from the org.omg.xmi.nsURI tag.

6.3.2 Overall Content Structure

```
2:ContentElements    ::= ( 3:ObjectAsElement )*
                         ( 6:ClassAttributes )*
                         ( 7:OtherLinks )*
```

2.	The contents are the XML representations of top level objects, classifier level attributes, and other links. The top level objects will include those that have a composite link with no reference from the composite to component.

6.3.3 Object Structure

```
3:ObjectAsElement    ::= "<" 3a:ObjectTagName 3c:ObjectAttribs ("/>")?
                             5:ObjectContents
                             3b:ObjectEndTag
3a:ObjectTagName     ::= 1g:Namespace // XMI name//
3b:ObjectEndTag      ::= ("</" 3a:ObjectTagName ">")?
3c:ObjectAttribs     ::= ( 1c:StartAttribs )?
                           3d:IdentityAttribs
                         ( 3f:TypeAttrib )?
                           3g:FeatureAttribs
3d:IdentityAttribs   ::= ( 3e:IdAttribName "=’" // id // "’")?
                         ( 1b:XMINamespace "label=’" //label// "’" )?
                         ( 1b:XMINamespace "uuid=’"  //uuid// "’" )?
3e:IdAttribName      ::= 1b:XMINamespace "id" | // id attrib name //
3f:TypeAttrib        ::= (1b:XMINamespace | 1g:Namespace)
                         "type=’"  3a:ObjectTagName "’"
3g:FeatureAttribs    ::= ( 3h:DataValueAttrib
                         | 3i:EnumValueAttrib
                         | 3j:RefValueAttrib )*
3h:DataValueAttrib   ::= 3l:AttribName "=’" //value// "’"
3i:EnumValueAttrib   ::= 3l:AttribName "=’" //enumeration literal// "’"
3j:RefValueAttrib    ::= 3l:AttribName "=’" 3k:RefValues "’"
3k:RefValues         ::= ( //reference id// " " )*
3l:AttribName        ::= // XMI name of attribute //
```

3.	An object has a starting element, contents, and a closing element. If the contents are empty, you may end the starting element with “/>.” You use this production rule to serialize top-level objects and to serialize objects that are the values of attributes and references. You may also use this production rule to serialize structured types. To serialize structured types, use the name of the structure rather than the class name, and use the attribute production rules to serialize the fields of the structure and their values.
	`<department xmi:id="Department_1"/>`

Example 6.1 - Instance of a class with empty contents

3a.	If the object is a top-level object, the tag name is the namespace name followed by “:” and the XMI name for the object. The XMI name for the object is either the name of the object’s class or the value of the org.omg.xmi.xmiName tag. If the object is the value of an attribute or reference, the XMI name is the name of the attribute or reference, or the value of the org.omg.xmi.xmiName tag. The namespace name is ignored for an object that is the value of an attribute or reference.
	`<complexco:department xmi:id="Department_1"/>`

Example 6.2 - Instance of a class, namespace name is its package name

3b.	The end tag name is the same as the start tag name, preceded with a “/.” An end tag need not be written if there is no content for the object.
3c.	The XML attributes for an object are the optional start attributes, identity attributes, and attributes corresponding to an object’s features (its attributes and references). The start attributes must be written if the object is a top-level object and it is not inside an XMI element specified by production rule 1a:XMI.

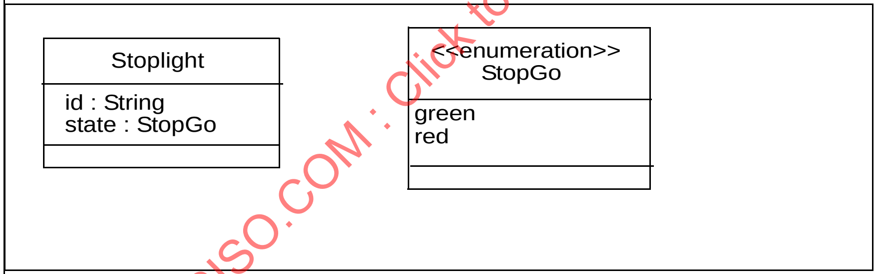
```
<Company xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
     xmi:id=Company_1" name="Acme">
         ..........
</Company>
```

Example 6.3 - Company is the top-level object in a document with no XMI element

3d.	The identity attributes consist of an optional id, label, and uuid. If the element has a MOF uuid, it may be used here.
3e.	By default, the name of the identity attribute is “id” in the XMI namespace. However, if an org.omg.xmi.idName tag has been specified, the name of the identity attribute is the value of that tag.
3f.	If the class of the object cannot be determined unambiguously from the model, you must specify the class name using the “type” attribute in either the XMI namespace or the schema instance namespace whose URI is “http://www.w3.org/2001/XMLSchema-instance”. The value of this attribute is defined by the XML Schema Part 1: Structures specification to be a QName, consisting of a namespaces name for the value’s class (if there is one and it is not the default namespace for the document), a “:”, and the name of the value’s class. Refer to the schema specification for more details. You may only use the XML schema instance type attribute if org.omg.xmi.useSchemaExtensions is true. 6.4.3, “Derived Types and References,” on page 62 provides an example of the use of the “type” attribute.

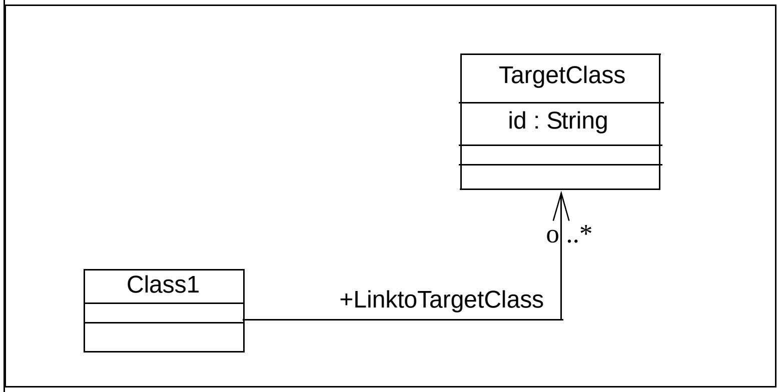
3g.	The XML attributes of the element correspond to attributes whose type is a data value or enumeration, or references whose values are objects in the document. You may not serialize an attribute or reference as both an XML element and an XML attribute in the same object. You must not serialize an attribute or reference as an XML attribute if the value of the org.omg.xmi.element tag is “true.” You must not serialize an attribute or reference at all if the value of the org.omg.xmi.serialize tag is “false.” You must not serialize a reference at all if the org.omg.xmi.remoteOnly tag is true and the reference has a value that is an object in the same XML document. You may serialize classifier-level attributes with an object.
3h.	Use this production rule to serialize an attribute whose type is not an object and whose value can be represented by a string. Multi-valued attributes cannot be serialized as XML attributes. If the attribute’s type is one of the types defined by the XML Schema Part 2: Datatypes specification, serialize the value as specified in that specification.
	<Department xmi:id="Department_1" **number="13"**/>

Example 6.4 -. Instance of a class with a single valued attribute

3i.	Use this production rule to serialize an attribute whose type is an enumeration and whose value is one of the legal enumeration literals. If the org.omg.xmi.xmiName is specified for the literal, the value of that tag should be used; otherwise, the name of the enumeration literal specified in the model is used.
	 Stoplight id : String state : StopGo <<enumeration>> StopGo green red
	<Stoplight xmi:id="Stoplight_6" id="SL06" **state="red"** />

Example 6.5 - Instance of a class with an enumerated attribute

3j. 3k.	Use this production rule to serialize references whose values are objects that are serialized in the same document. The value of the XML attribute contains the XMI ID of each referenced object, separated by a space.



```
<Class1 xmi:id="Class1_1" LinktoTargetClass="TargetClass_1 TargetClass_2"/>
<TargetClass xmi:id="TargetClass_1" id="TC1 instance"/>
<TargetClass xmi:id="TargetClass_2" id="TC2 instance"/>
```

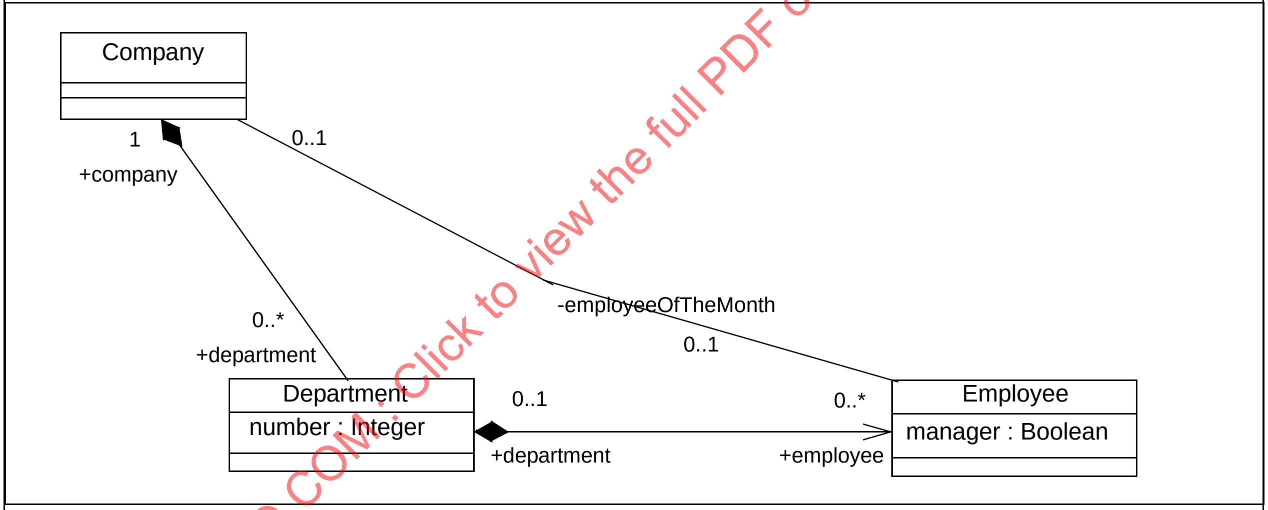
Example 6.6 - Association from an instance of a class to instances of another class

3l.	The name of the XML attribute is the name of the model attribute or reference, or the value of the org.omg.xmi.xmiName tag for the attribute or reference.

6.3.4 References

```
4:ReferenceElement ::= "<" 3a:ObjectTagName
                       ( 1c:StartAttribs )?
                       3d:IdentityAttribs
                       ( 3f:TypeAttrib )?
                       4a:LinkAttribs
                       "/>"
4a:LinkAttribs     ::= 1b:XMINamespace "idref=’" //reference id// "’"
                       | 4b:Link
4b:Link            ::= "href='" //URI reference// "'"
```

4.	Use this production rule to serialize a reference to an object using an XML element. If you use identity attributes, the values of the identity attributes must match the values of the identity attributes for the object that is referenced.
4a.	Use the idref attribute to specify the id of an XML element that is referenced in the document; use the href attribute to specify an XML element in another document. If the org.omg.xmi.href tag is “true,” you must not use the idref attribute; use the href attribute for references within the document and across documents.
4b.	An XMI link. The value of the href attribute is a URI reference that refers to an XML element in another document or in the same document. For example, if the href is “file:someFile.xmi#someId,” the href refers to an XML element in the “someFile.xmi” document whose XMI ID is “someId.” If the href is “#anotherId,” the href attribute refers to an XML element whose XMI ID is “anotherId” in the same document. XLinks are also supported in XMI. See 4.10.2, “Linking,” on page 20 for more information. See the W3C XLink and XPointer specification for production rules.



Document **CompanyKey_1.xml** contains a link to external document **CompanyKey_2.xml** for the **employeeOfTheMonth** association:

```
   <Company xmi:id="Company_1" name="Acme">
        <employeeOfTheMonth href="CompanyKey_2.xml#Employee_1" />
   </Company>
```

Document **CompanyKey_2.xml** contains the target of the link, and link back to original document:

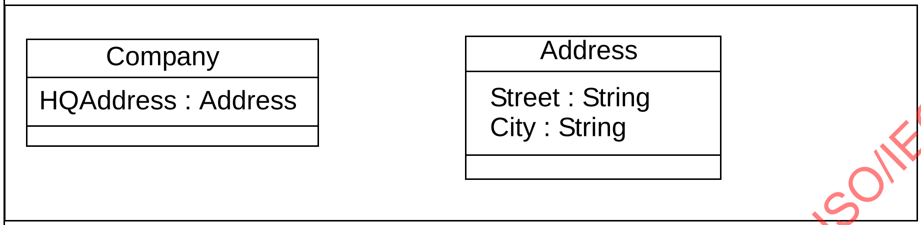
```
   <Employee xmi:id="Employee_1" name="Fatale, Natasha">
        <company href="CompanyKey_1.xml#Company_1" />
```

Example 6.7 - Linking across documents

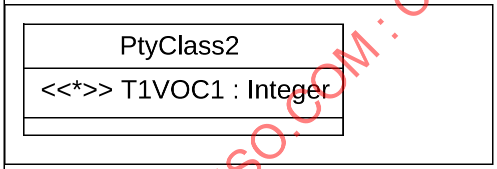
6.3.5 Object Contents

```
5:ObjectContents      ::= ( 5a:AttributeAsElmt
                          | 5h:ReferenceAsElmt>
                          | 5i:CompositeAsElmt )*
                          ( 5j:Extension )*
5a.AttributeAsElmt   ::=  ( 5b:AttribValueAsElement ) *
                          | 5f:NullValue
5b:AttribValueAsElmt ::= 3:ObjectAsElement
                          | 4:ReferenceElement
                          | 5c:DataValue
                          | 5d:EnumLiteral
5c:DataValue          ::= "<" 5e:AttribTagName ">"
                          //value//
                          "</" 5e:AttribTagName ">"
5d:EnumLiteral        ::= "<" 5e:AttribTagName ">"
                          //enumeration literal//
                          "</" 5e:AttribTagName ">"
5e:AttribTagName      ::= // XMI name for attribute//
5f:NullValue          ::= "<" 5e:AttribTagName 5g:NullAttrib "/>"
5g:NullAttrib         ::= 1g:Namespace "nil='true'"
5h:ReferenceAsElmt    ::= 4:ReferenceElement
5i:CompositeAsElmt    ::= 3:ObjectAsElement
5j:Extension          ::= "<" 1b:XMINamespace "extension"
                             (" extender='" // extender // "'")?
                             (" extenderID='" // extenderID // "'")?
                          ">"
                             // Extension elements //
                                "</" 1b:XMINamespace "extension>"
```

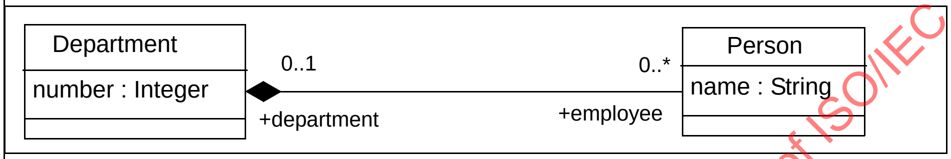
5.	The contents of an object are the attributes, references, and compositions that are serialized as XML elements, as well as the extensions. Note that 'contents' (component objects that are reached via composite links) without a composite reference are not subject to this production rule and so not written as nested elements: instead they are written as top-level elements. Any particular reference or single-valued attribute may be expressed as an XML element or XML attribute, but not both. You can specify whether an attribute or reference is serialized as an XML element or an XML attribute by using the org.omg.xmi.element tag. If the value of the org.omg.xmi.superClassFirst tag is "true," you must serialize inherited attributes, references, and compositions first, beginning at the top of the class hierarchy.

5a.	Each value of an attribute is represented by an XML element; for multi-valued attributes, there is one XML element for each value. Null values may be serialized as well, unless the value of the org.omg.xmi.includeNils tag is "false," in which case you may not serialize null values.
5b. 5c.	If the attribute value is an object, it is serialized using the 3:ObjectAsElement production rule unless the object is in another document, in which case the 4:ReferenceElement production rule is used.
	 Company HQAddress : Address Address Street : String City : String
	<Company xmi:id="Company_1" name="Acme"> **<HQAddress xmi:id="Address_1" Street="Side Street"** </Company>**City="Hometown"/>**

Example 6.8 - Value of attribute HQAddress is an object

	Use this production rule to save values of attributes that are neither objects nor enumeration literals. If the type of the attribute is one of the types defined by the XML Schema Part 2: Datatypes specification, the value must be serialized according to that specification.
	
	<PtyClass2 xmi:id="PtyClass2_1"> **<T1V0C1>1001</T1V0C1>** **<T1V0C1>2001</T1V0C1>** </PtyClass2>

Example 6.9 - Multi-valued attribute, with each value serialized as an element

5d.	The enumeration literal is either the name of the literal from the model or the value of the org.omg.xmi.xmiName tag.
5e.	The XMI name for the attribute is either the name of the attribute from the model or the value of the org.omg.xmi.xmiName tag.
5g.	The null attribute has the name “nil” in a namespace whose URI is “http://www.w3.org/2001/XMLSchema-instance.”
5i.	Use this production to serialize composite relationships as elements.  Department number : Integer 0..1 +department 0..* +employee Person name : String <Department xmi:id="Department_1" number="13"> **<employee xmi:id="Employee_2" name ="Glozic, Dejan" />** **<employee xmi:id="Employee_3" name ="Andrews, Gilbert" />** **<employee xmi:id="Employee_4" name ="Beisiegel, Gloria" />** </Department>

Example 6.10 - Aggregation serialized as elements

5j.	Each extension element has an optional extender and extenderID attribute; its content can be anything.

6.3.6 Packages

```
6:Package           ::= "<" 6a:PackageTagName 3c:ObjectAttribs ">"
                             (7:ClassAttributes | 8:OtherLinks)*
                        "</" 6a:PackageTagName ">"
6a:PackageTagName ::= 1g:Namespace //XMI name//
```

6.	This element is only serialized if there are classifier-level attributes that have not been serialized in objects, or other links that have not been serialized with objects, either.

6.3.7 Attributes

```
7:ClassAttributes    ::= ( 5a:AttributeAsElmt )*
```

7.	All classifier-level attributes are expressed using the XML element form, unless they have already been serialized in objects.

6.3.8 Other Types of Links

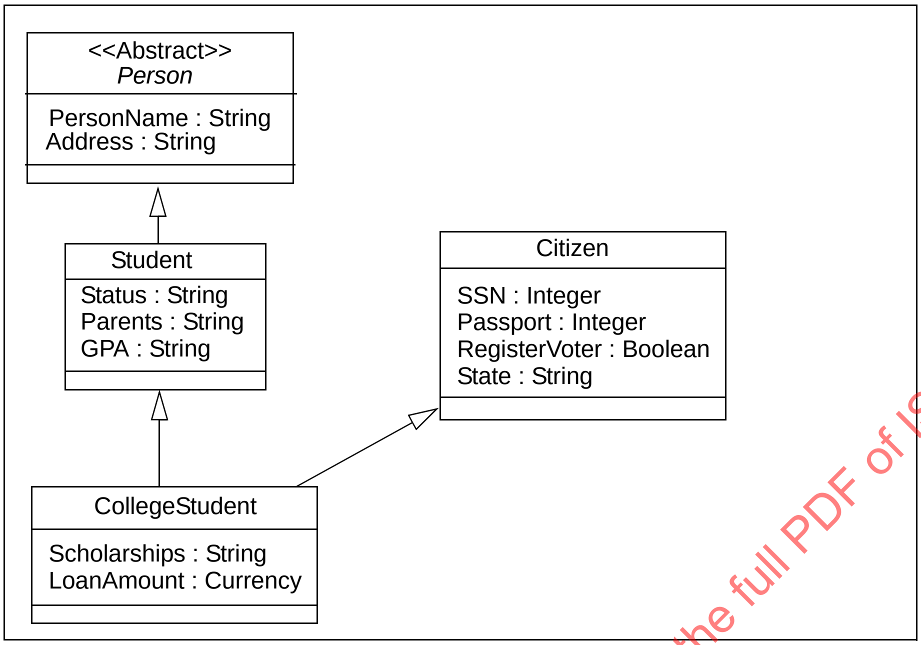
```
8:OtherLinks          ::= "<" 8a:AssocTagName 3c:ObjectAttribs ">"
                            ( 8b:AssociationEndRef 8b:AssociationEndRef )*
                           "</" 8a:AssocTagName ">"
8a:AssocTagName       ::= //XMI name for the association//
8b:AssociationEndRef  ::= 4:ReferenceElement
```

8.	All associations that have no references are placed here. Each associationEnd’s links are contained as pairs of nested XML elements.
8a.	The tag name of the association is the name of the association specified in the model or the value of the org.omg.xmi.xmiName tag.
8b.	A reference to the linked element from the AssociationEnd; the tag name of the referenced element should be the XMI name for the association end, which is either the name of the association end specified in the model or the value of the org.omg.xmi.xmiName tag.

6.4 Additional Examples

6.4.1 Inheritance

Attributes and associations are inherited from parent classes. For example, in the model below, CollegeStudent inherits directly from Student and Citizen, and indirectly from Person.

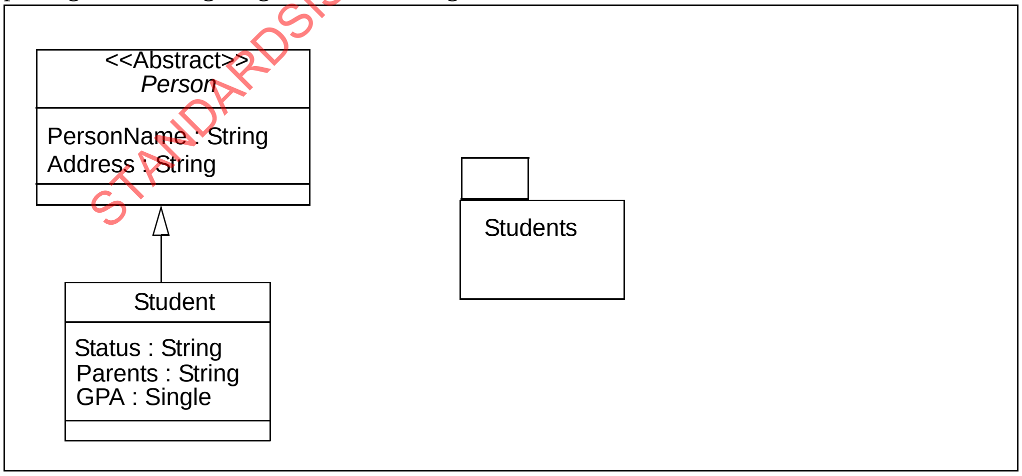


An instance of CollegeStudent can include attributes inherited from each of these parent classes:

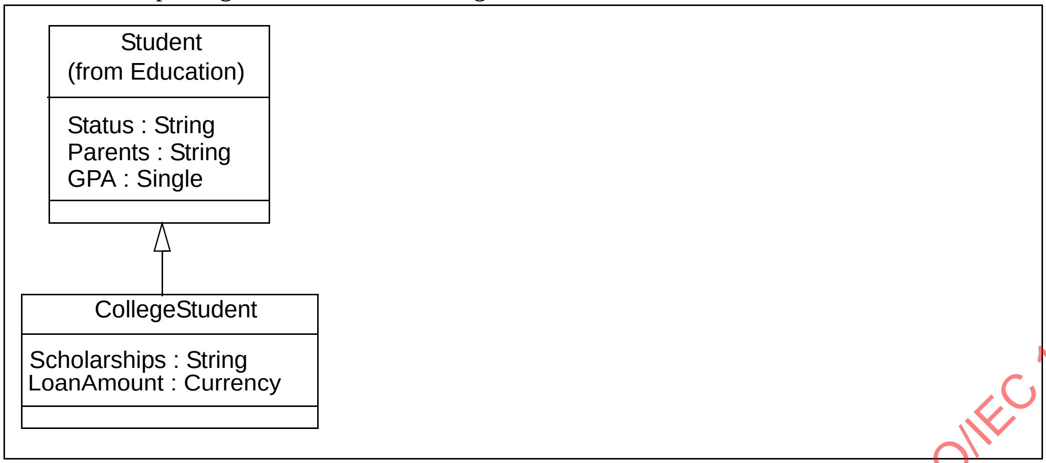
```
<CollegeStudent xmi:id="CollegeStudent_1"
        PersonName="Andrew Pham" GPA="4.95" SSN="1234567890" />
```

6.4.2 Nested Packages

The following model shows the **Education** package, which contains another package called **Students**, where the Students package has an org.omg.xmi.nsPrefix tag set to “Students:”



The Students package contains class CollegeStudent:

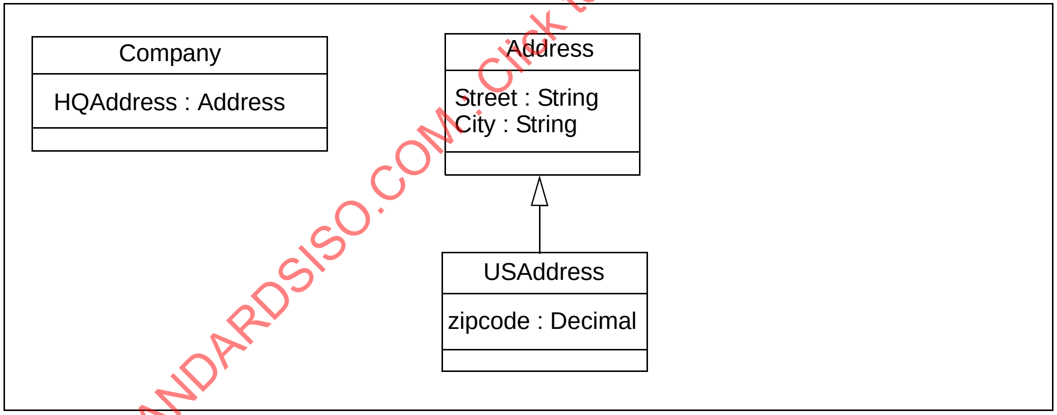


The package nesting can be expressed in the qualifier for the CollegeStudent element:

```
<Students:CollegeStudent xmi:id="CollegeStudent_1"
              PersonName="Andrew Pham" GPA="4.95" SSN="1234567890" />
```

6.4.3 Derived Types and References

In the following example, class Company has attribute HQAddress whose type is another class, the Address class:



Address has a subclass, USAddress. An instance of Company can use xsi:type to indicate that its HQAddress is actually of type USAddress and includes a zipcode:

```
<Company xmi:id="Company_1" name="Acme">
     <HQAddress xmi:type="USAddress" xmi:id="Address_1"
            Street="Side Street" City="Hometown" zipcode="90210"
```

Similarly, if a model contains a reference to a class that has a subclass, xmi:type can be used in an instance to indicate that the reference is actually to the subclass.

6.5 Document Production Rules for Non-Primitive Data

MOF 1.4 added a set of non-primitive data types. The document production rules for these data types are defined using the existing production rules in 6.3, "EBNF Rules Representation," on page 49. They are described in the following sub clauses.

6.5.1 Structure Type

The document production rules for a structure type with structure fields are the same as for a class with attributes. The production rules for classes are defined starting with 3:ObjectAsElement. For structure types, use the structure type's name instead of class name, and the structure field names instead of attribute names.

Example: ST is a structure type with structure fields sf1 (type String) and sf2 (type A). A is a class with attributes a1 (type String) and a2 (type String). An instance of ST is serialized as:

```
<ST xmi:id="ST_1" sf1="xxxx">
   <sf2 xmi:id="sf2_1" xmi:type="A" a1="yyyy" a2="zzzz"/>
</ST>
```

6.5.2 Enumeration Type

The document production rules for enumeration types are defined by rules 3i:EnumValueAttrib and 5d:EnumLiteral.

6.5.3 Alias Type

The document production rules for an alias type are the same as for its base type, but using the alias type's name instead of the base type's name.

6.5.4 Collection Type

The document production rules for a collection type are the same as for a class that has one attribute with the same type and multiplicity as the collection type. The production rules for classes are defined starting with 3:ObjectAsElement. For collection types, use the collection type name instead of class name. Use the collection type's type name instead of attribute name.

Example: CT is a collection type. CT's type is A, and it has multiplicity 0..*. A is a class with attributes a1 (type String) and a2 (type String). An instance of CT of size 3 is serialized as:

```
<CT xmi:id="CT_1">
   <A xmi:id="A_1" a1="string" a2="another string"/>
   <A xmi:id="A_2" a1="stuff" a2="more stuff"/>
   <A xmi:id="A_3" a1="xxxx" a2="yyyy"/>
</CT>
```

(Blank page)

7 Production of MOF from XML

7.1 Introduction

XML is increasingly becoming an information source, supplementing existing sources such as analysis (UML), software (Java, C++), components (EJB, IDL, Corba Component Model), and databases (CWM). Although XML does not define objects, it can be used as an input source of true object definitions by supplementing the XML with additional information or conventions.

This Clause describes the following algorithms for producing object definitions in MOF from XML input sources:

- DTD to MOF production
- XML to MOF production
- XML Schema to MOF production

This sub clause describes mappings to produce MOF declarations from XML documents, DTDs, and XML schemas. The mappings are not unique since XML-only forms of information are not rich enough to produce an unambiguous MOF representation.

These mechanisms are not necessary for reading XMI documents, since XMI is rich enough to interchange complete MOF information without loss or ambiguity.

The approach in these productions has been to provide reverse mappings for only the most common declarations used in XML. The productions are in two parts: rules and parameterized mappings. Each of the three XML information sources has its own rule to extract the corresponding class and attribute declarations they represent. The parameterized mappings are MOF rules to produce the simplest MOF classes and attributes with specific parameters that may be customized by an implementation that has additional domain knowledge beyond the production inputs.

7.2 DTD to MOF

When a DTD is used to create a MOF metamodel, the DTD is read declaration by declaration, and MOF definitions are added accordingly. For each type of declaration, one of the following MOF definitions is added by following the particular rule. The mapping may be customized by setting the parameters in the second table.

As an example, this DTD would by default produce these MOF declarations:

```
DTD:
<!ELEMENT Car (Engine, Door*)>
<!ATTLIST Car make CDATA #IMPLIED model CDATA #IMPLIED>
<!ELEMENT Engine (#PCDATA)>
<!ELEMENT Door EMPTY>
<!ATTLIST Door side CDATA #REQUIRED>

MOF:
Class Car {
   Attribute make : String;
   Attribute model : String;
   Association engine : Engine 1..* containment one-way;
   Association door : Door   1..* containment one-way;
}
```

```
Class Engine {
   Attribute value : String 0..1;
}

Class Door {
   Attribute side : String 0..1;
}
```

Rule	DTD Declaration	MOF Definition
1	<!ELEMENT E>	Class E with Supertype (E).
2	<!ATTLIST E A Type Occurs>	Attribute named A of Class E with type AttributeType(E, A, Type) and multiplicity AttributeMult(E, A, Occurs).
3	<!ELEMENT E (F)>	TypedElement(E,F) Attribute or Association to Class F and name RoleName(E, F).
4	<!ELEMENT E (#PCDATA)>	Attribute named TextName(E) of type AttributeType(E, TextName(E)).
5	<!ELEMENT E ANY>	TypedElement(E, “ANY”) Attribute or Association to Supertype(“Any”) and name RoleName(E, “ANY”).

Parameters	Defaults
Supertype(Element name)	Node
AttributeType (Element name, Attribute name, Type name) AttributeNult (Element name, Attribute name, Occurs style) TypedElement (Element name, TypedElementname)	String for Type CDATA Lookup MOF type for IDREF 0..1 Association: containment by value, multiplicity 0..*, one way navigable, Attribute: multiplicity 0..*
RoleName (Element name, TypedElement name)	LowerCase TypedElement name
TextName(Element name)	“value”

7.3 XML to MOF

When an XML document has no additional type information, it is possible to generalize to produce a minimal MOF representation. The mapping uses the same optional parameters as the DTD to MOF mapping.

The processing of the generalization follows these steps:

1. Parse the XML document into a DOM tree.
2. Select an existing MOF metamodel or create an empty MOF metamodel.
3. Perform a depth-first traversal of the XML document’s DOM tree. At each node, apply the appropriate generalization operation from the table, based on the type of parent and child nodes encountered.

This is an example result from mapping from an XML document to MOF:

```
XML:
<Car make="Ford" model="Mustang">
   <Engine>240 HP</Engine>
   <Door side="left"/>
   <Door side="right"/>
</Car>

MOF:
Class Car {
   Attribute make : String;
   Attribute model : String;
   Association engine : Engine  1..* containment one-way;
   Association door : Door   1..* containment one-way;
}

Class Engine {
   Attribute value : String  0..1;
}

Class Door {
   Attribute side : String  0..1;
}
```

Rule	DOM Parent Node	DOM Child Node	MOF Definition
1	Element E	None	Class E with Supertype(E)
2	Element E	Attribute A	Attribute named A of Class E with type AttributeType(E,A, "CDATA") and multiplicity AttribiteMult(E,A, "#IMPLIED")
4	Element E	Element F	TypedElement(E,F) Attribute or Association to Class F and name RoleName(E,F)
5	Element E	Text, CharacterData, or CDATASection	Attribute named TextName(E) of type AttributeType(E, TextName(E))

7.4 XML Schema to MOF

The following subset of the example of XML Schema, representing a portion of the purchase order example of the XML Schema specification, part 0, mapped to MOF using the reverse engineering table below.

The processing follows these steps:

1. The XML Schema is parsed.
2. Schema declarations corresponding to one of the three rules are processed while traversing the XML Schema depth-first.

XML Schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

 <xsd:element name="comment" type="xsd:string"/>

 <xsd:complexType name="PurchaseOrderType">
   <xsd:sequence>
     <xsd:element name="shipTo" type="USAddress"/>
     <xsd:element name="billTo" type="USAddress"/>
     <xsd:element ref="comment" minOccurs="0"/>
    </xsd:sequence>
   <xsd:attribute name="orderDate" type="xsd:date"/>
 </xsd:complexType>

 <xsd:complexType name="USAddress">
   <xsd:sequence>
     <xsd:element name="name"   type="xsd:string"/>
     <xsd:element name="street" type="xsd:string"/>
     <xsd:element name="city"   type="xsd:string"/>
     <xsd:element name="state"  type="xsd:string"/>
     <xsd:element name="zip"    type="xsd:decimal"/>
   </xsd:sequence>
   <xsd:attribute name="country" type="xsd:NMTOKEN"
       fixed="US"/>
 </xsd:complexType>

</xsd:schema>
```

MOF:

```
<Class name="PuchaseOrder">
                <attribute name="shipTo" type="USAddress"/>
                <attribute name="billTo" type="USAddress"/>
                <attribute name="comment" type="mof:String" multiplicty="0..1"/>
                <attribute name="orderDate" type="mof:String"/>
</Class>
<Class name="USAddress">
                <attribute name="name" type="mof:String"/>
                <attribute name="street" type="mof:String"/>
                <attribute name="city" type="mof:String"/>
                <attribute name="state" type="mof:String"/>
                <attribute name="zip" type="mof:Integer"/>
                <attribute name="country" type="mof:String"/>
</Class>
```

Data types map to “mof:String” unless defined in the user model, except “xsd:decimal” and its restrictions map to “mof:Integer” and “xsd:boolean” maps to “mof:boolean.”

Rule	**XML Schema**	**MOF Definition**
1	Element(E), ComplexType(E), SimpleType(E) with base (S)	Class E with Supertype(S)
2	Sequence(L), List(L), Choice(L) containing Rule 1 (E2) and minOccurs(min), MaxOccurs(max)	Attribute E2 of AttributeType (E, L, E2) with multiplicity min..max
3	Attribute(A) Type(T)	Attribute A with AttributeType(E, A, T)

(Blank page)

8 XML Schema Model

8.1 Introduction

This sub clause describes the MOF model for XML Schema declarations using UML notation. The model is a straightforward mapping from the XML Schema specification, where classes in the model have a direct correspondence to a definition in XML Schema. This definition assumes a strong working knowledge of XML Schema and refers throughout to the XML Schema specification for the detailed description of constructs that are defined by XML Schema.

8.2 XML Schema Structures

This model corresponds to the structures defined in the XML Schema Part 1, Structures.

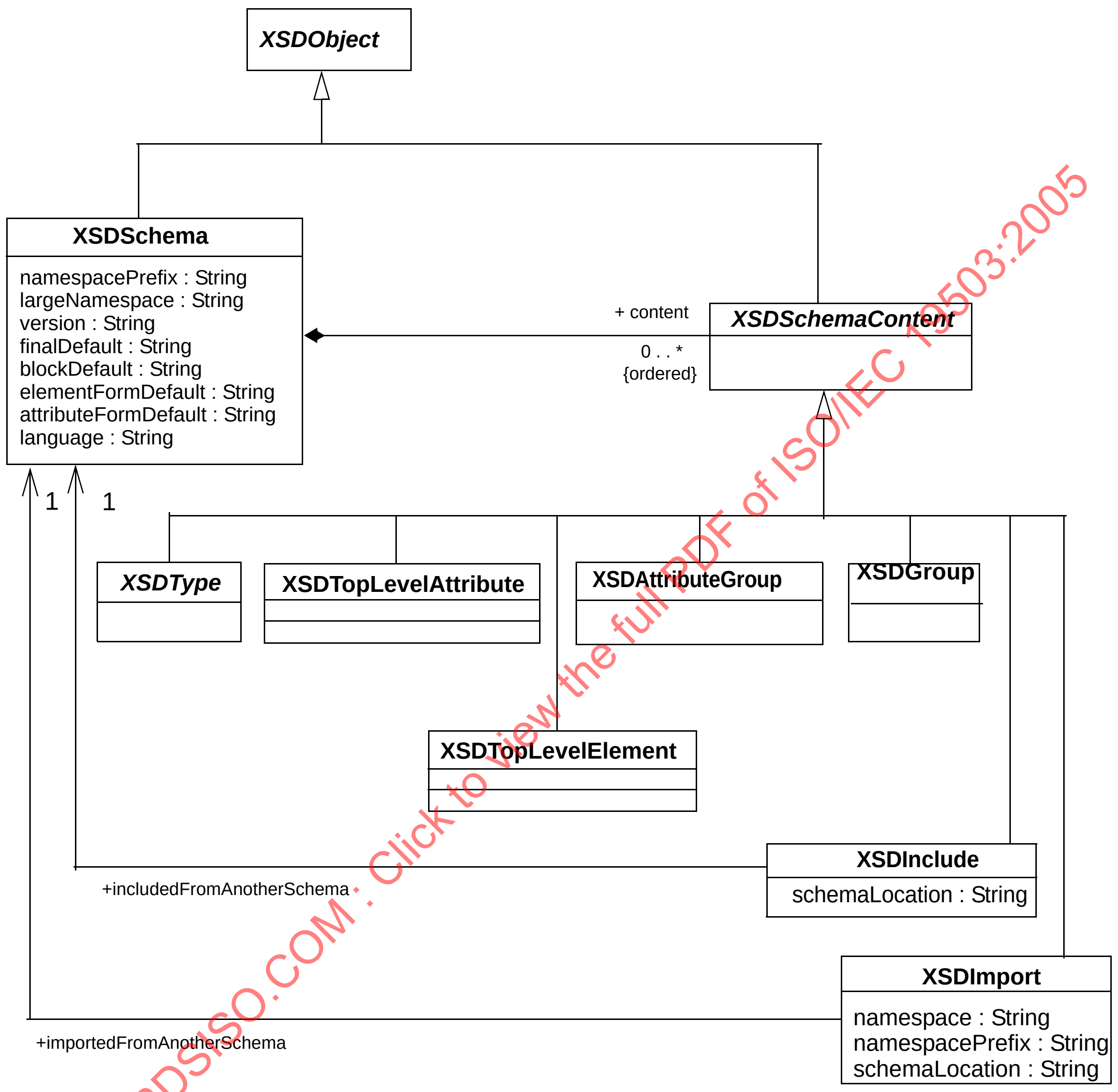


Figure 8.1 - XML Schema top level declarations

The top level XML Schema declarations consist of the description of the schema itself (namespace prefix, target namespace, etc.) and the declarations within the schema. These declarations include global scope Attributes, global scope Elements, attribute groups, type declarations (extending from XSDGroup), and imports from other schemas.

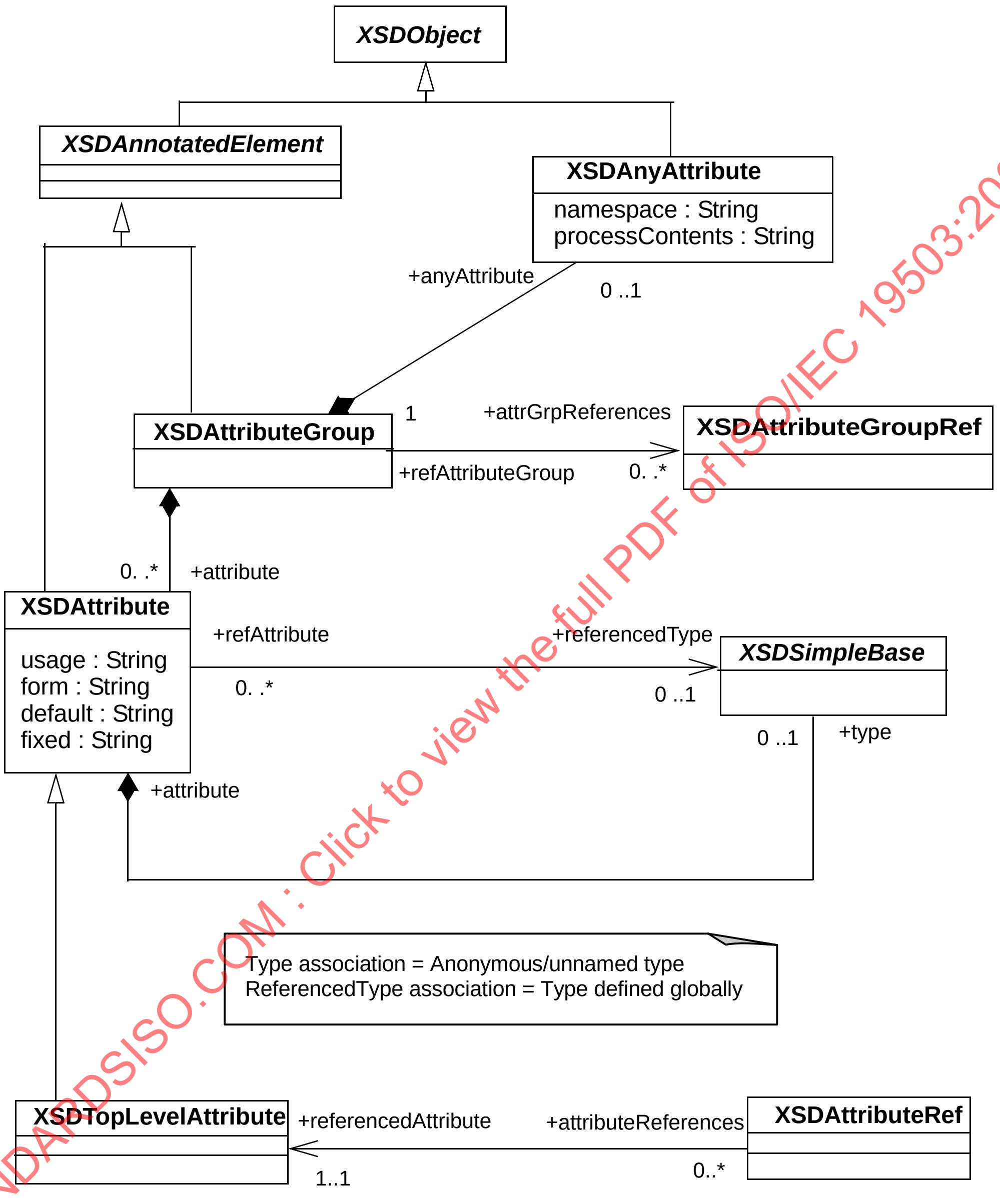


Figure 8.2 - XML Schema Attribute Declarations

An XML attribute has a name inherited from XSDNamedElement, and a simple type that is either defined within its scope or referred to externally. The attribute may be annotated.

The attribute may be defined within an attribute group for reuse later. Attribute groups may refer to other attribute groups.

A top level attribute may be referred to by other attribute uses.

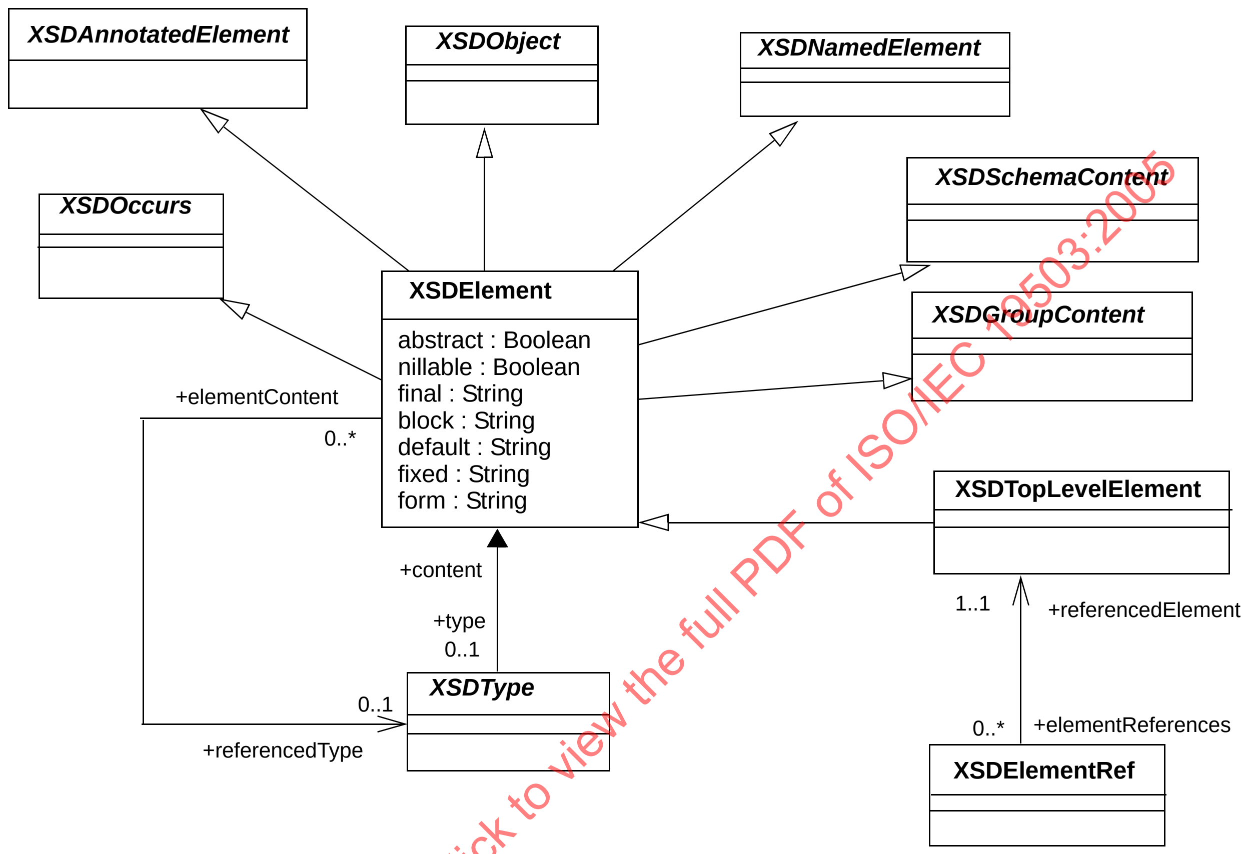


Figure 8.3 - XML Schema Element declaration

An Element declaration includes a name from XSDNamedElement, an annotation from XSDAnnotatedElement, and may be used as content for a schema or a group.

The element may define new types in its own declaration or refer to types declared elsewhere.

A top level element declaration may be referred to by element references.

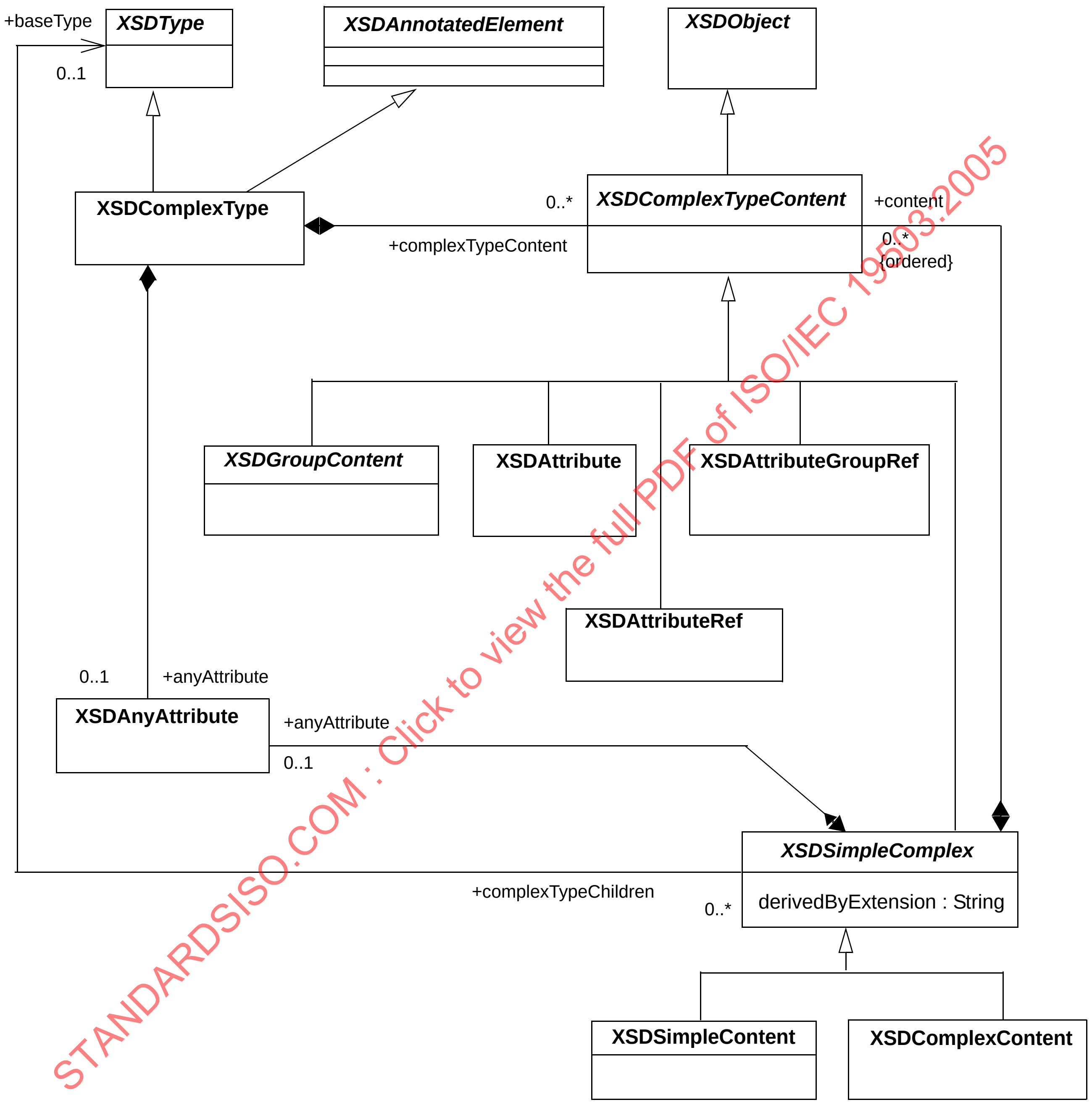


Figure 8.4 - XML Schema Complex type declaration

A complex type is both a type and an annotated element. The complex type has complexTypeContent that may be a group of types and declared simple or complex types. The type may have attributes, or refer to attributes or attribute groups.

Complex type contents may be derived by extension or restriction, and may be simple or complex.

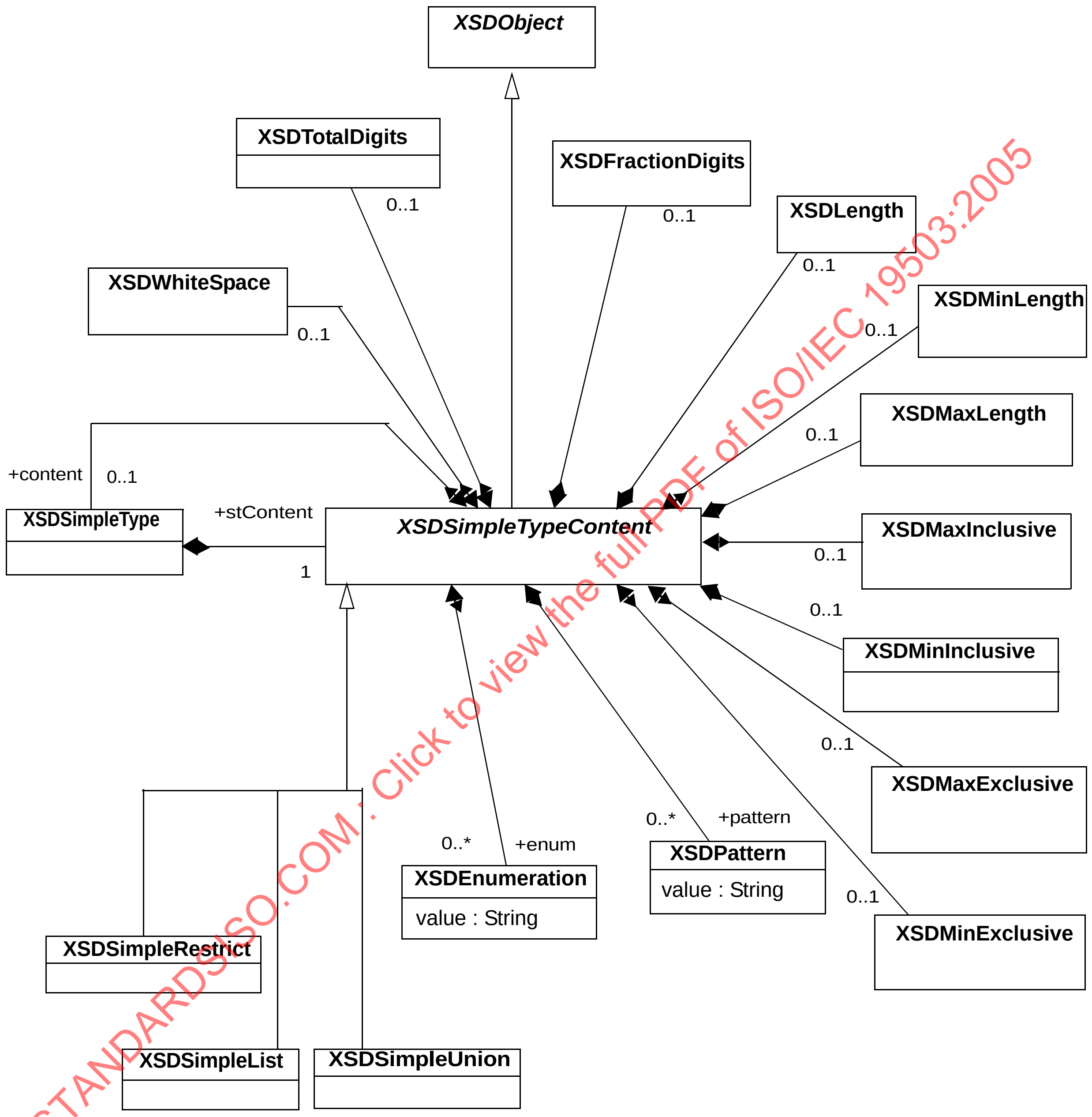


Figure 8.5 - XML Schema Simple type content declarations

The content of a simple type is described in terms of facets. These facets include white space, digit representation, length, ranges, patterns, enumerations, unions, and lists.

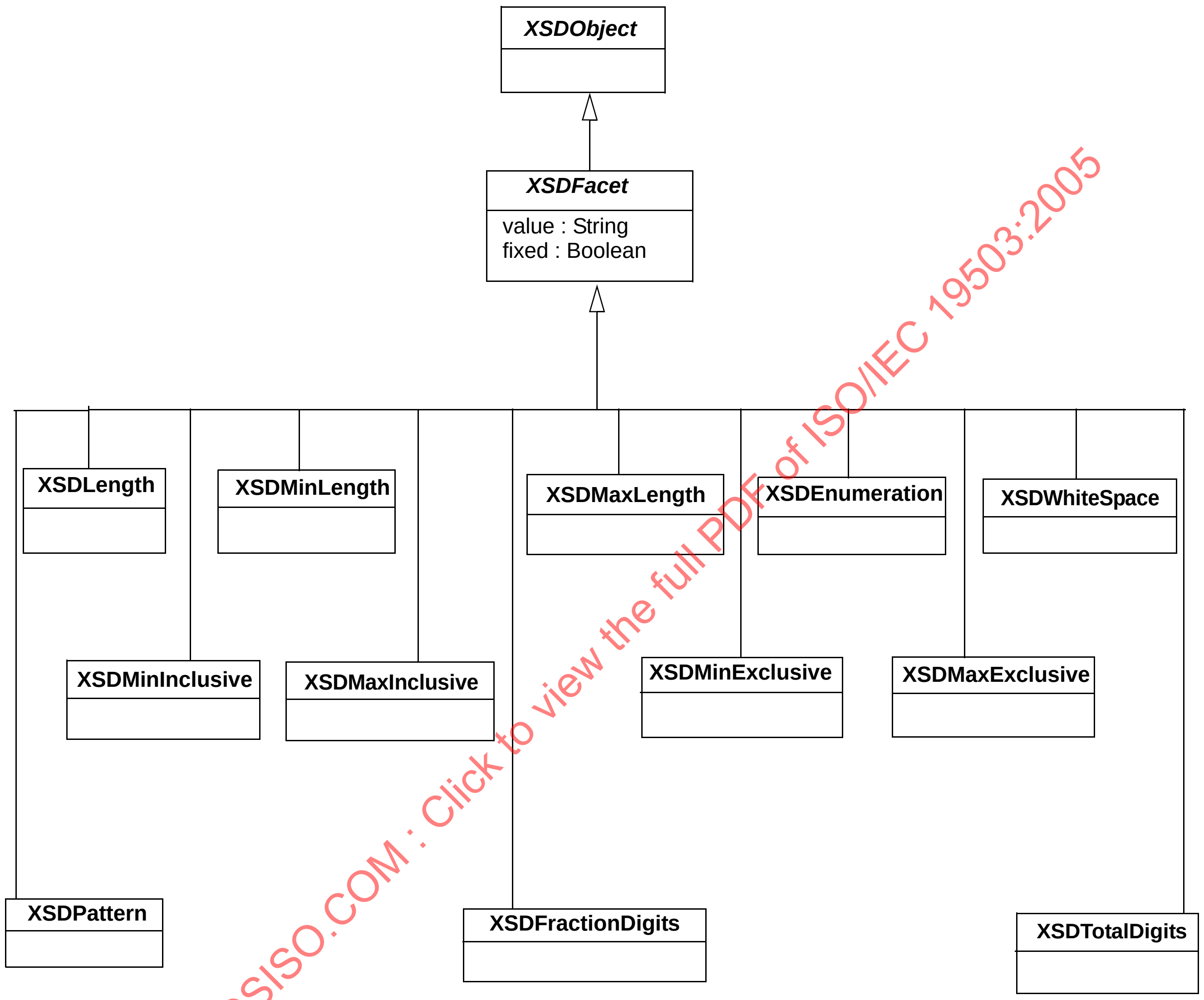


Figure 8.6 - XML Schema Facets

There are many types of facets used in simple type content declarations. They share a common root, XSDFacet, an abstract class that declares the value of the facet, and if the facet is fixed.

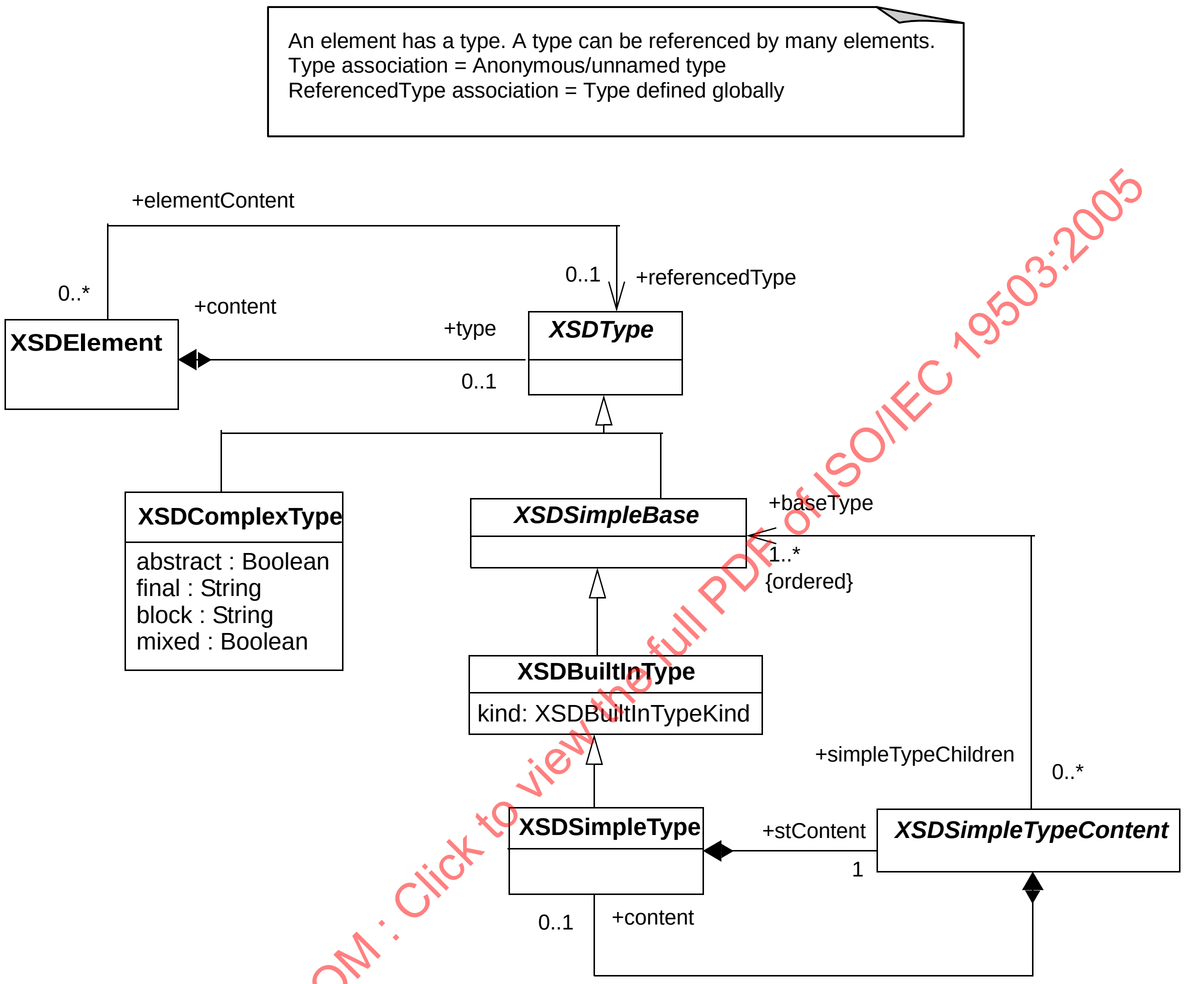


Figure 8.7 - XML Schema Type declaration

An XML Schema Type may be declared in a schema or within an element. The type may be a simple or complex type. Simple types may be one of the built-in, predefined types from XML Schema part 2, data types, or they may be a user-defined simple type.

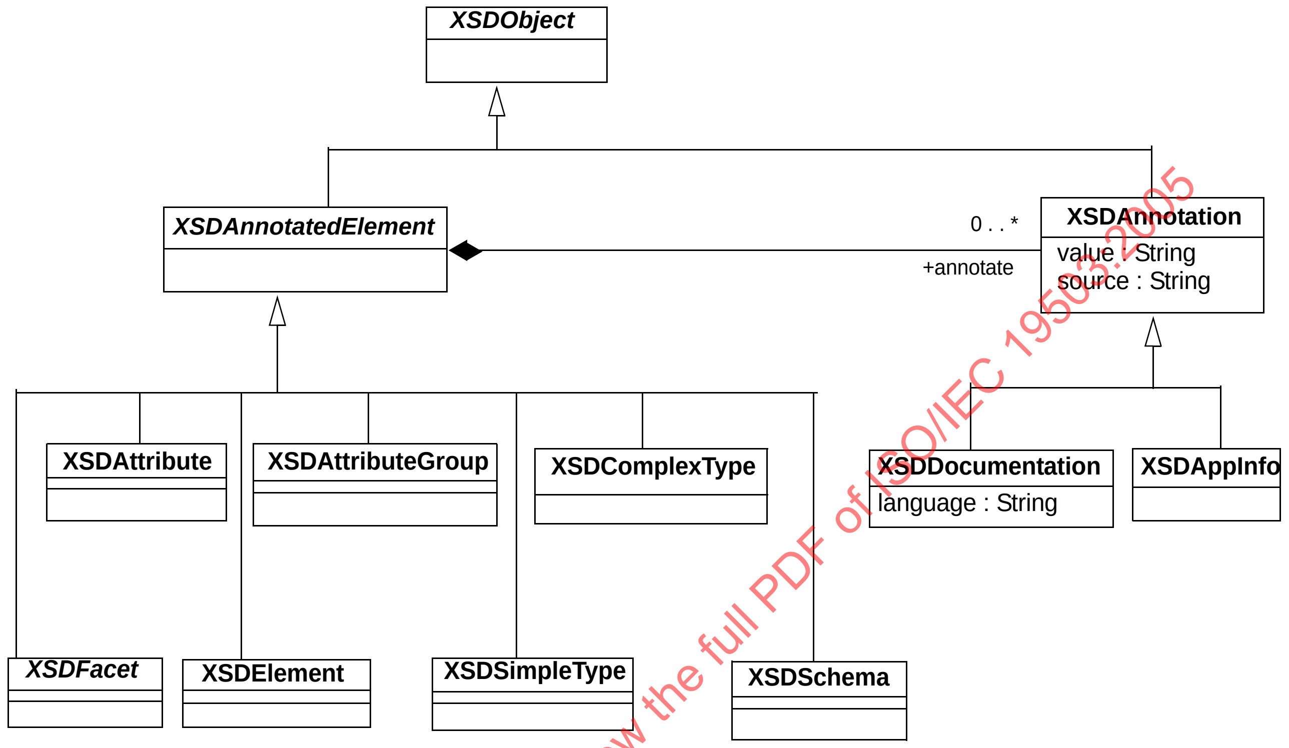


Figure 8.8 - XML Schema annotated elements

Many XML Schema declarations may contain annotations. These elements are attributes, attribute groups, elements, simple and complex types, facets, and schemas. An annotation may include documentation or application information.

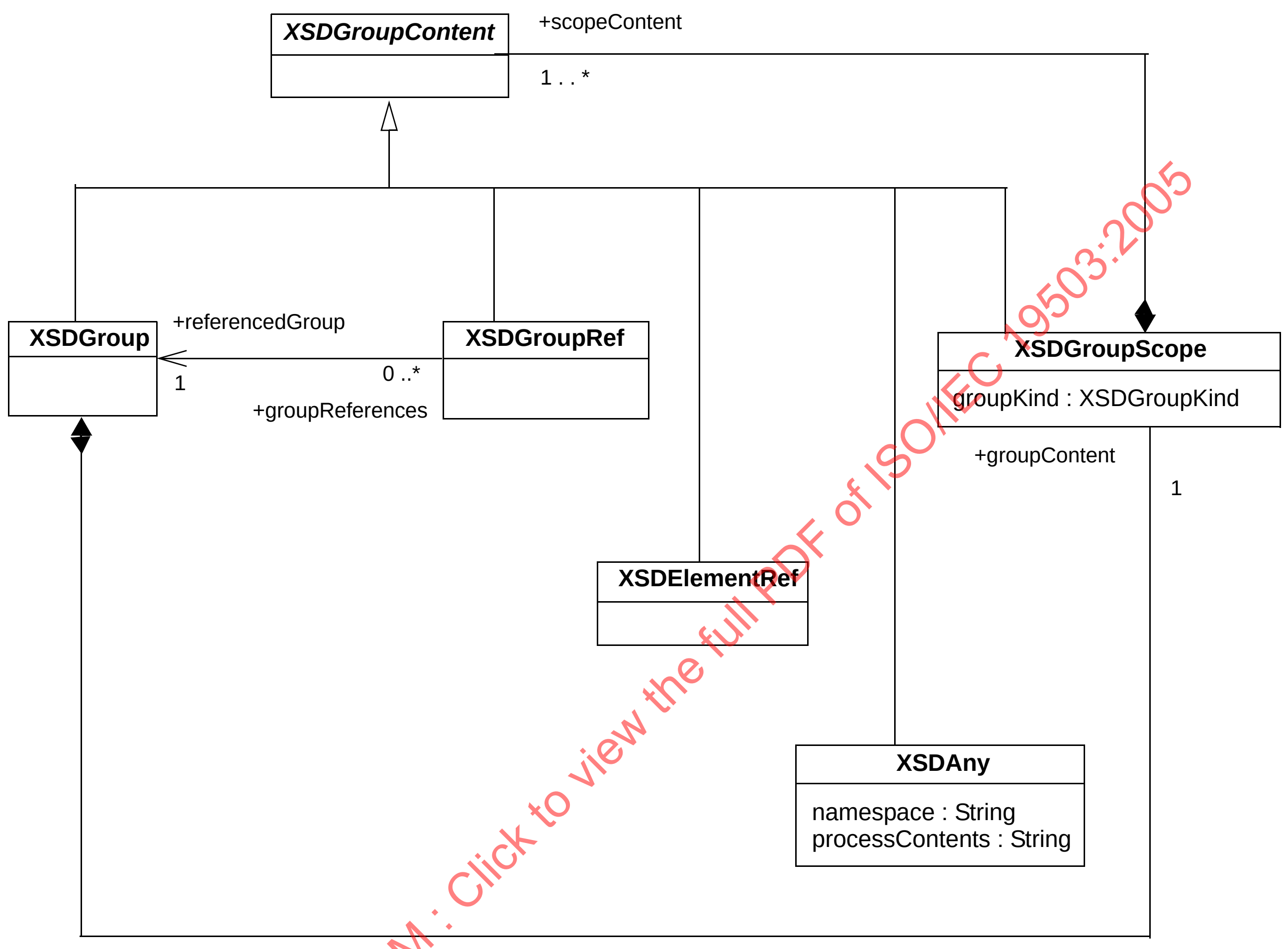


Figure 8.9 - XML Schema group declarations

A group may contain other groups, references to other groups or elements, or contain declarations of additional groups and elements in terms of choice, sequence, or all. Groups with Any content may also be declared.

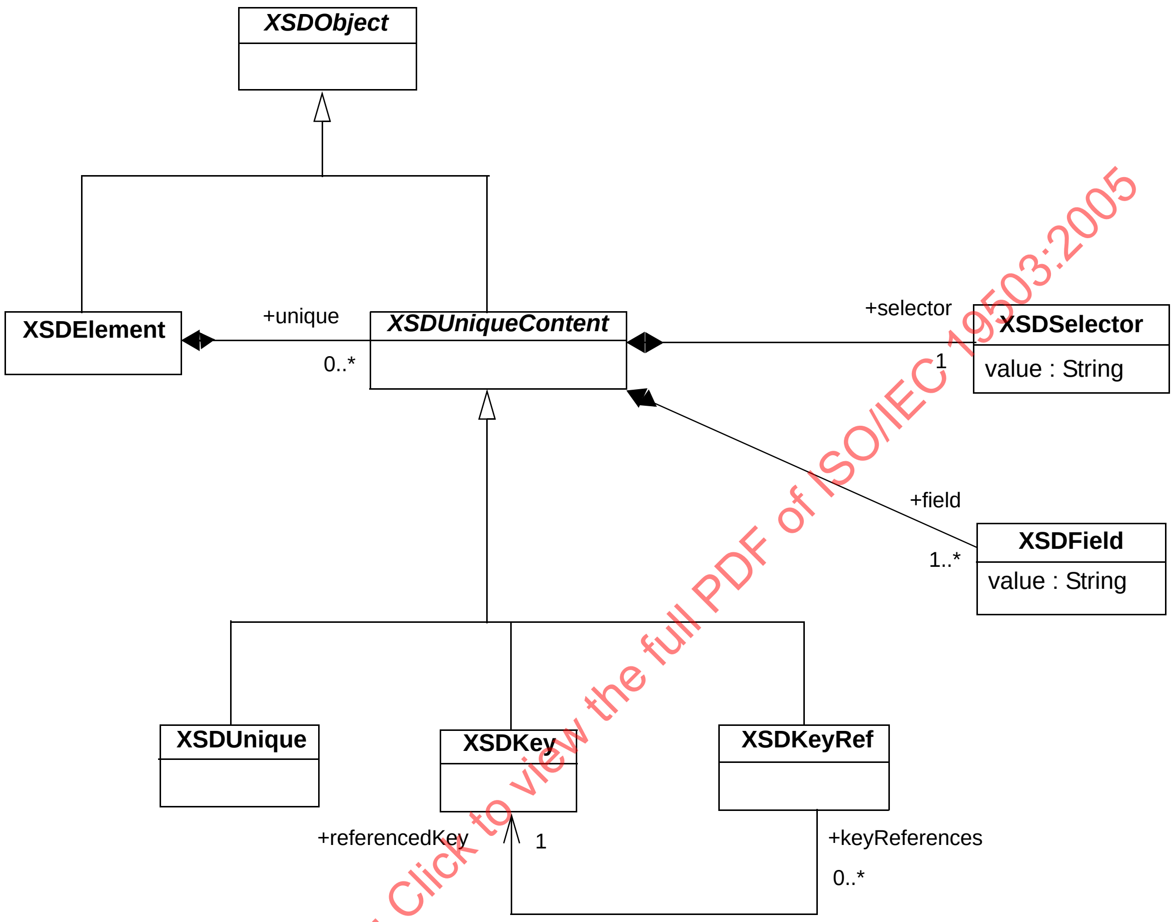


Figure 8.10 - XML Schema key declaration

A key declaration is made based on the uniqueness of the content of an element. The elements contents are measured based on selections on its attributes. Keys may refer to other keys.

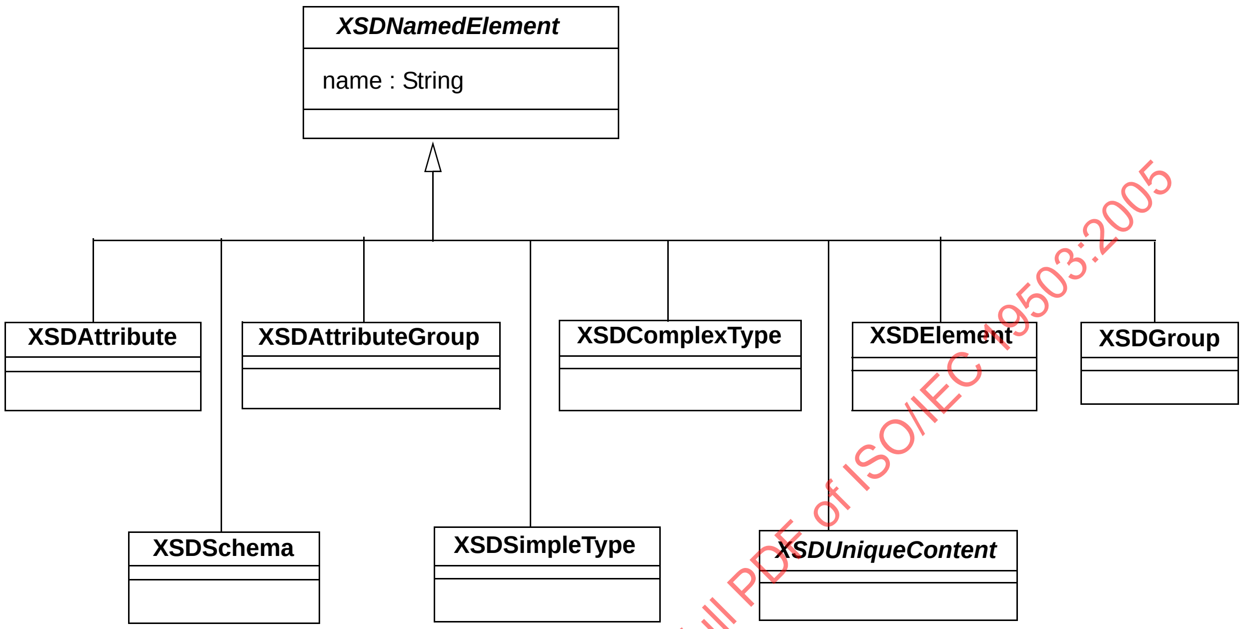


Figure 8.11 - XML Schema name declarations

Attributes, attribute groups, elements, simple and complex types, groups, unique content, and schemas are named.

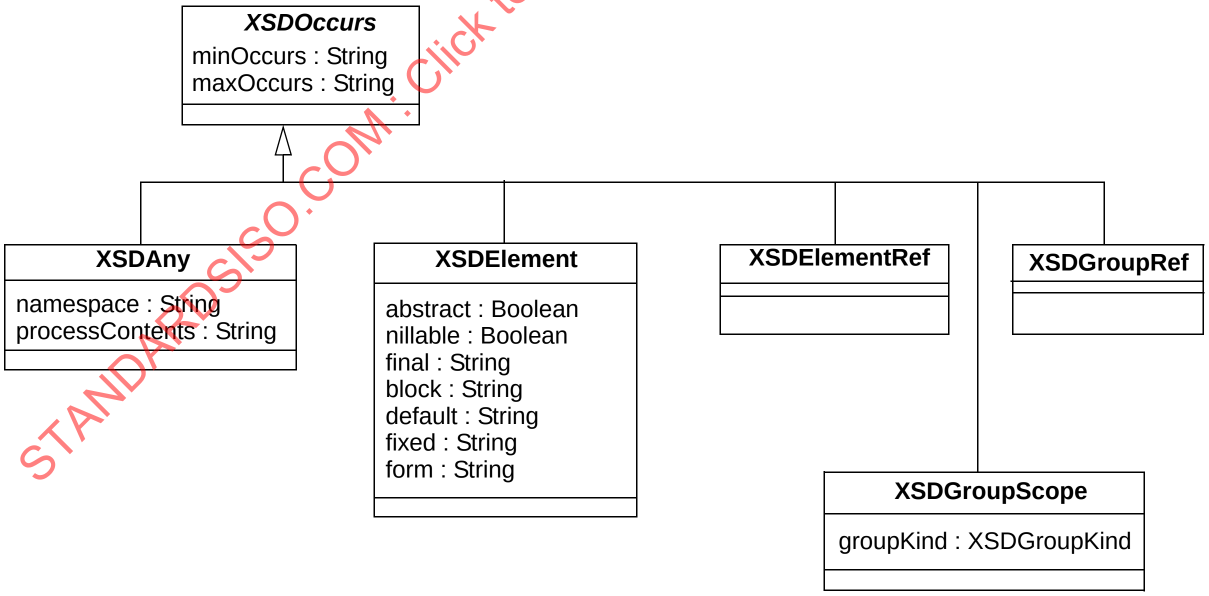


Figure 8.12 - XML Schema occurrence particles

The occurrence particle in declarations of elements, element references, anys, groups, and group references is factored into the Occurs abstract class.

8.2.1 XSDSchema

XSDSchema is an XML Schema Declaration.

Extends: XSDObject, XSDNamedElement, XSDAnnotatedElement

Attributes:
- namespacePrefix : String
- targetNamespace : String
- version : String
- finalDefault : String
- blockDefault : String
- elementFormDefault : String
- attributeFormDefault : String
- language : String

8.2.2 XSDAttribute

An XML Schema attribute declaration.

Extends: XSDComplexTypeContent, XSDNamedElement, XSDAnnotatedElement

Attributes:
- usage : String
- form : String
- default : String
- fixed : String

8.2.3 XSDElementRef

A reference to an XML Schema element declaration.

Extends: XSDGroupContent, XSDOccurs

8.2.4 XSDAttributeGroup

An XML Schema attribute group declaration.

Extends: XSDSchemaContent, XSDNamedElement, XSDAnnotatedElement

8.2.5 XSDAttributeGroupRef

A reference to an attribute group.

Extends: XSDComplexTypeContent

8.2.6 XSDType

An XML Schema abstract type.

Extends: XSDSchemaContent

8.2.7 XSDBuiltInType

An XML Schema predefined datatype.

Extends: XSDSimpleBase

8.2.8 XSDComplexType

A ComplexType can derive from another Complex Type or another Simple Type. Complex types may have substantial structure.

Extends: XSDType, XSDNamedElement, XSDAnnotatedElement

Attributes:
abstract : Boolean
final : String
block : String
mixed : Boolean

8.2.9 XSDComplexTypeContent

The content of an XML Schema.

Extends: XSDObject

8.2.10 XSDSchemaContent

The content of an XML Schema.

Extends: XSDObject

8.2.11 XSDElement

An XML Schema element declaration.

Extends: XSDObject, XSDNamedElement, XSDOccurs, XSDAnnotatedElement, XSDGroupContent, XSDSchemaContent.

Attributes:
abstract : Boolean
nullable : Boolean
final : String
block : String
default : String
fixed : String
form : String

8.2.12 XSDSimpleBase

An abstract base class for XML Schema simple types.

Extends: XSDType

8.2.13 XSDPattern

A pattern constraint on a datatype.

Extends: XSDFacet

Attributes:
 value : String

8.2.14 XSDEnumeration

An enumeration constraint on a datatype.

Extends: XSDFacet

Attributes:
 value : String

8.2.15 XSDInclude

An XML Schema include declaration.

Extends: XSDSchemaContent

Attributes:
 schemaLocation : String

8.2.16 XSDImport

An XML Schema import declaration.

Extends: XSDSchemaContent

Attributes:
 namespace : String
 namespacePrefix : String
 schemaLocation : String

8.2.17 XSDGroup

An XML Schema group declaration.

Extends: XSDSchemaContent, XSDGroupContent, XSDNamedElement

8.2.18 XSDGroupKind

Declares whether the groups contents will be one of each of its contents, a choice of one of its contents, or a sequence of all of its contents.

Enumeration literals:
 all
 choice
 sequence

8.2.19 XSDGroupScope

A nested XML Schema group declaration that may be declared as all, choice, or sequence.

Extends: XSDGroupContent, XSDOccurs

Attributes:
 groupKind : XSDGroupKind

8.2.20 XSDGroupContent

An abstract class representing contents of an XML Schema group declaration.

Extends: XSDComplexTypeContent

8.2.21 XSDGroupRef

A reference to an XML Schema group declaration.

Extends: XSDGroupContent, XSDOccurs

8.2.22 XSDKey

The declaration of a Key.

Extends: XSDUniqueContent

8.2.23 XSDKeyRef

A reference to the declaration of a key.

Extends: XSDUniqueContent

8.2.24 XSDUnique

The concrete declaration of the unique fields.

Extends: XSDUniqueContent

8.2.25 XSDUniqueContent

The type of content that is uniquely keyed.

Extends: XSDObject, XSDNamedElement

8.2.26 XSDSelector

The selector of an XML Schema uniqueness declaration.

Attributes:
  value : String

8.2.27 XSDField

The fields to apply the selector of an XML Schema uniqueness declaration.

Attributes:
  value : String

8.2.28 XSDObject

XSDObject in an abstract superclass to facilitate modeling of XML Schema.

8.2.29 XSDAnnotatedElement

XSDAnnotatedElement is an abstract class for XML Schema constructs that may be annotated.

Extends: XSDObject

8.2.30 XSDDocumentation

XSD documentation is the documentation of an XML Schema construct.

Extends: XSDAnnotation

Attributes:
  language : String

8.2.31 XSDAppInfo

Provides application specific information.

Extends: XSDAnnotation

8.2.32 XSDAnnotation

An XML Schema annotation.

Extends: XSDObject

Attributes:
value : String
source : String

8.2.33 XSDSimpleContent

XML Schema declaration of the content of a simple type.

Extends: XSDSimpleComplex

8.2.34 XSDComplexContent

XML Schema declaration of the content of a simple type.

Extends: XSDSimpleComplex

8.2.35 XSDSimpleComplex

XML Schema extended simple or complex types. Types may be extended by extension or restriction.

Extends: XSDComplexTypeContent

Attributes:
derivedByExtension : Boolean

8.2.36 XSDSimpleTypeContent

The declaration of simple type contents.

Extends: XSDObject

8.2.37 XSDSimpleRestrict

A simple type restriction.

Extends: XSDSimpleTypeContent

8.2.38 XSDSimpleList

A simple type list.

Extends: XSDSimpleTypeContent

8.2.39 XSDSimpleUnion

A simple type union.

Extends: XSDSimpleTypeContent

8.2.40 XSDSimpleType

An XML Schema simple type declaration. Simple types have minimal structure.

Extends: XSDBuiltInType, XSDNamedElement, XSDAnnotatedElement

8.2.41 XSDFacet

XML Schema type declarations use a series of facets to define the particular behavior. The XSDFacet is an abstract class that is specialized by the type of facet.

Extends: XSDObject, XSDAnnotatedElement

Attributes:
- value : String
- fixed : Boolean

8.2.42 XSDLength

The length facet.

Extends: XSDFacet

8.2.43 XSDMinLength

The minLength facet.

Extends: XSDFacet

8.2.44 XSDMaxLength

The maxLength facet.

Extends: XSDFacet

8.2.45 XSDMinInclusive

The minInclusive facet.

Extends: XSDFacet

8.2.46 XSDMaxInclusive

The maxInclusive facet.

Extends: XSDFacet

8.2.47 XSDMinExclusive

The minExclusive facet.

Extends: XSDFacet

8.2.48 XSDMaxExclusive

The maxExclusive facet.

Extends: XSDFacet

8.2.49 XSDTotalDigits

The totalDigits facet.

Extends: XSDFacet

8.2.50 XSDFractionDigits

The fractionDigits facet.

Extends: XSDFacet

8.2.51 XSDWhiteSpace

The whiteSpacefacet.

Extends: XSDFacet

8.2.52 XSDAny

The Any content for an XML Schema group content declaration.

Extends: XSDGroupContent, XSDOccurs

Attributes:
 namespace : String
 processContents : String

8.2.53 XSDAnyAttribute

The XML Schema reference to any attributes with non-schema namespace.

Extends: XSDObject

Attributes:
 namespace : String
 processContents : String