

Aeronautical Information Exchange Model (AIXM)

AIXM UML to XML Schema Mapping

Copyright: 2006 - EUROCONTROL and Federal Aviation Administration

All rights reserved.

This document and/or its content can be download, printed and copied in whole or in part, provided that the above copyright notice and this condition is retained for each such copy.

For all inquiries, please contact:

Brett BRUNK - brett.brunk@faa.gov

Eduard POROSNICU - eduard.porosnicu@eurocontrol.int

Edition No.	Issue Date	Author	Reason for Change
0.1	2006/06/22	Brett Brunk	First Edition
	2006/08/25	Barb Cordell	Incorporate data modelling section
	2006/09/06	Vamshi Reddy	Incorporate Data type section
	2007/01/30	Barb cordell	Update for Release Candidate 1

CONTENTS

1	SCOPE	1
1.1	Introduction	1
1.2	Background.....	1
1.3	Glossary.....	1
1.4	References	1
2	AIXM UML MODELING.....	1
2.1	AIXM is GML	1
2.2	Features, Objects, Properties and Groups.....	1
2.2.1	Features.....	2
2.2.2	Objects	2
2.2.3	Properties	2
2.2.3.1	Simple properties	2
2.2.3.2	Association Cardinality	3
2.2.3.3	Specialization/Inheritance	4
2.2.4	modelGroups.....	4
3	MAPPING TO AIXM GML SCHEMA.....	5
3.1.1	Abstract and Standard AIXM XML Schema elements.....	5
3.1.1.1	ObstaclePropertyGroup.....	6
3.1.1.2	ObstacleTimeSliceType	9
3.1.1.3	ObstacleTimeSlice	10
3.1.1.4	ObstacleTimeSlicePropertyType.....	11
3.1.1.5	ObstacleType	11
3.1.1.6	Obstacle.....	12
3.1.1.7	ObstaclePropertyType.....	12
3.1.1.8	ObstacleArrayPropertyType, ObstacleArray and ObstacleArrayType	13
3.1.1.9	ObstacleExtension.....	13
3.1.2	Objects	14
3.1.2.1	ObstaclePartExtension	14
3.1.2.2	ObstaclePartPropertyGroup	15
3.1.2.3	ObstaclePartType	16
3.1.2.4	ObstaclePart	17
3.1.2.5	ObstaclePartPropertyType	17
3.1.2.6	ObstaclePartArrayType.....	18
3.1.3	modelGroups.....	18
3.1.4	Relationships to Objects	19
3.1.5	Relationships to Features	21
3.2	Data Types	22
3.3	Unit of Measurement	24

1 Scope

1.1 Introduction

The purpose of this document is to describe how the AIXM Conceptual Model is converted into the AIXM XML Schema. The UML to XML Schema conversion is illustrated using a series of examples from the AIXM 5 XML schema.

1.2 Background

With version 5, AIXM is transitioning from Entity-Relationship modeling to Unified Modeling Language (UML) class modeling. The AIXM Conceptual Model and data standard are maintained as a UML model.

1.3 Glossary

See AIXM proposal document.

1.4 References

1. Geographic Information – Spatial Schema. ISO 19107. First Edition, 2003-05-01
2. Geography Markup Language (GML). ISO/TC 211/WG 4/PT 19136 OGC GML RWG. Committee Draft. 2004-02-07.
3. UML 2.0 In a Nutshell. Dan Pilone. O'Reilly Media Inc. 2005.
4. AIXM 5 Modelling Conventions.
5. AIXM 5 Proposal
6. AIXM 5 Profile of GML
7. Galdos
8. Rational Edge UML Basics

2 AIXM UML Modeling

2.1 AIXM is GML

As discussed extensively in the AIXM 5 Proposal [6], the AIXM exchange model is an XML exchange standard based on a subset of GML [6]. Essentially:

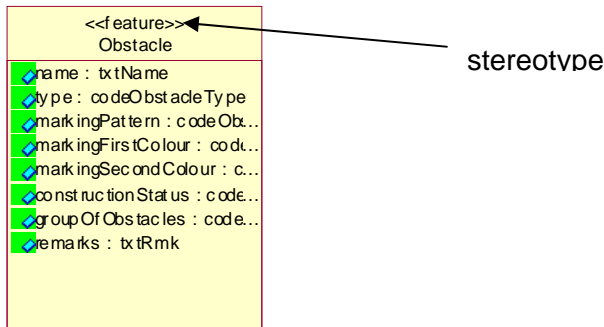
- AIXM Features are GML Features.
- AIXM Objects are GML objects.
- AIXM follows the GML Object-Property concept

2.2 Features, Objects, Properties and Groups

In AIXM, UML Features, Objects and Groups are represented as UML classes distinguished by their stereotypes. Stereotypes are used to further define and extend standard UML concepts. The stereotypes used in the AIXM 5.0 model are defined in the AIXM 5 Proposal.

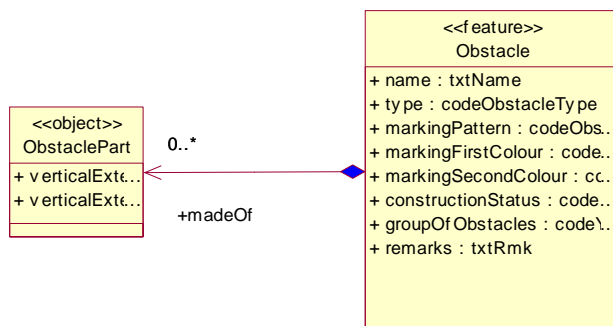
2.2.1 Features

Features describe real world entities and are the fundamental objects used in GML. GML features can be concrete and tangible, or abstract and conceptual and can change in time [7]. Features are represented as classes with a stereotype <<feature>>. We encode AIXM features as GML dynamic features with an array of Timeslice objects that describe the temporal changes of the AIXM feature. Timeslice objects are discussed extensively in the AIXM design document.



2.2.2 Objects

In AIXM objects are abstractions of real world entities that do not exist outside of a feature. Whenever a property has a multiplicity greater than one and cannot exist outside the feature these properties are represented as an object with the proper UML composite relationship as shown. We encode AIXM objects as GML objects.



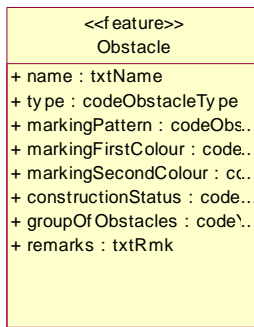
2.2.3 Properties

Properties are the attributes and relationships of a feature or object that are used to characterize the object. In the UML attributes are used to describe simple properties of a feature, object or group. Relationships are used to describe associations to features, objects and groups. Whenever a property has a multiplicity greater than one it is described using a UML relationship with cardinality.

2.2.3.1 Simple properties

Simple properties of cardinality one are shown as attributes in the UML diagram and they are XML encoded as optional elements in the class group.

To illustrate, the Obstacle feature has several simple properties:

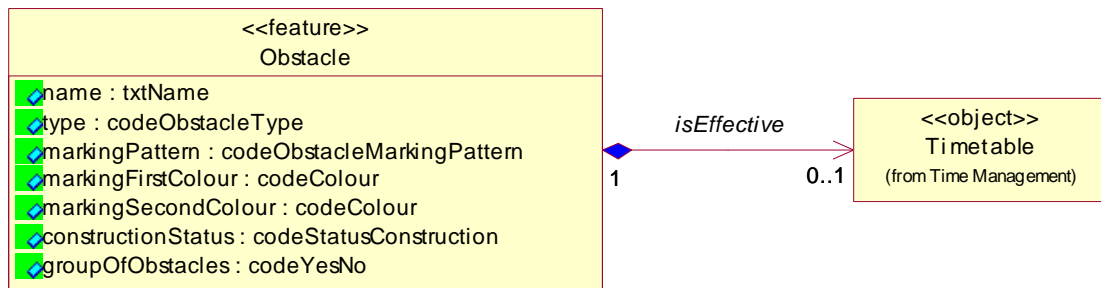


The **name** and **type** are simple properties of the **Obstacle** feature. Note that the Feature/Objects are written in UpperCamelCase and the properties are in lowerCamelCase.

2.2.3.2 Association Cardinality

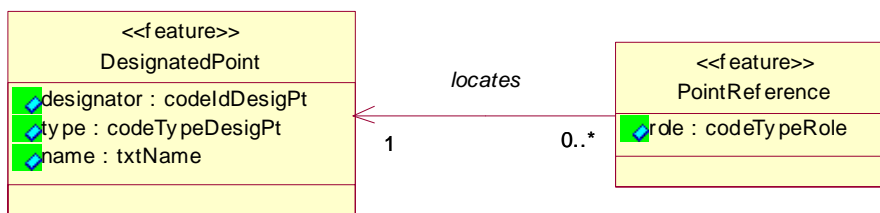
2.2.3.2.1 Relationships to Objects

Relationships to objects are depicted by the standard UML composition (*aggregation by value*) association.

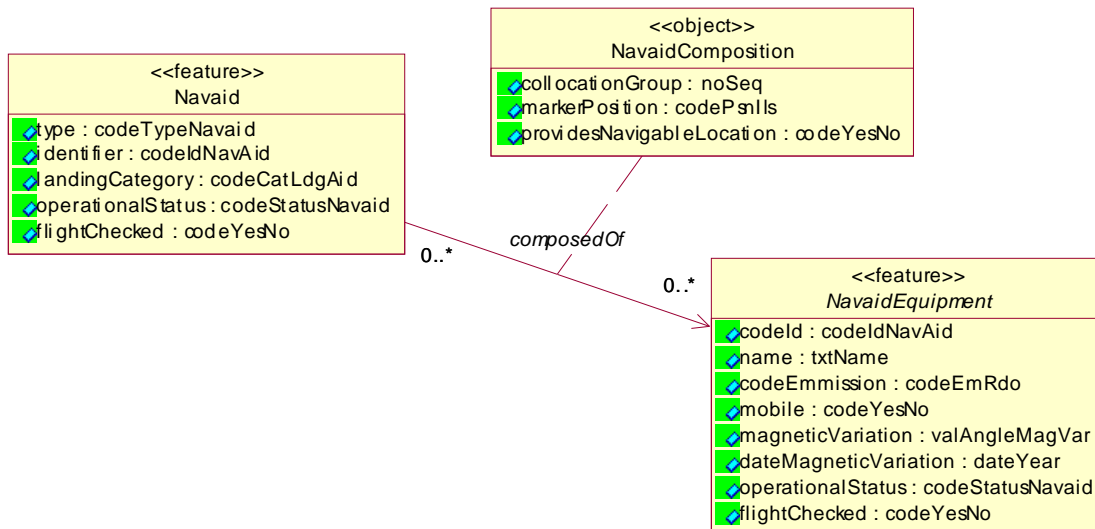


2.2.3.2.2 Relationships to Features

In AIXM Relationships to features are described with a uni-directional standard UML association. A uni-directional association shows that two classes are related, but only one class knows that the relationship exists [8]. In the example below the Point Reference feature knows about the Designated Point but the Designated Point does not know about the Point Reference. This uni-directional association is important when the UML model is translated to XSD.



When we need to include information about a relationship we create a standard UML association class. The class is attached to the relationship. Again the uni-directional arrow is important when mapping to the XSD.

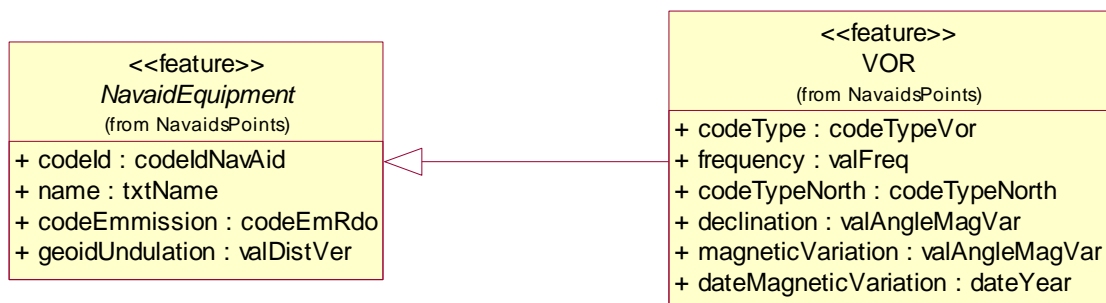


2.2.3.3 Specialization/Inheritance

Inheritance refers to the ability of one class (specialized or child class) to inherit the properties of another class (generalized or parent class), and then add new properties of its own. Features must only inherit from other features. Objects must only inherit from other objects. Multiple inheritance is not allowed.

Within our XML schema, specialization implies two characteristics:

1. Substitutability. The more general feature or object can be substituted by a specialization. In the XML schema this is supported using substitution groups.
2. Property inheritance. The specialized feature inherits all of the properties of the more general feature. In the XML schema including the properties of the general class into the specialized class supports this.

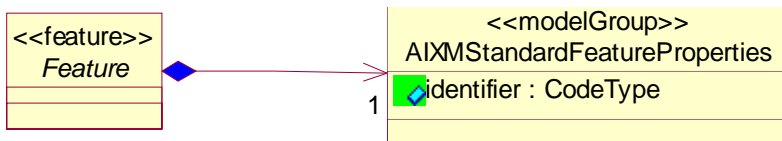


In this example the VOR is a specialized kind of NavaidEquipment.

2.2.4 modelGroups

In AIXM modelGroups are collections of common properties that will be included in one or more feature or group. We encode AIXM modelGroups as XSD groups containing the common properties.

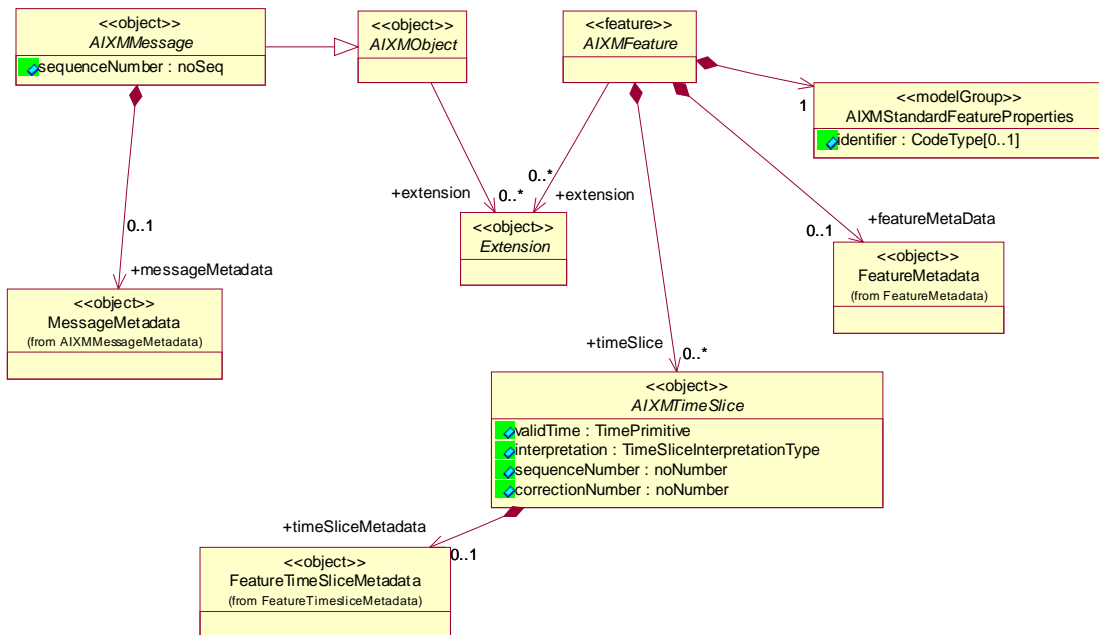
For example the AIXMStandardFeatureProperties modelGroup contains an identifier. Every feature in AIXM (GML) will have the identifier property. :



3 Mapping to AIXM GML Schema

3.1.1 Abstract and Standard AIXM XML Schema elements

The AIXM-Abstract.xsd schema contains a set of abstract AIXM elements, complex types and groups that are used as the building blocks for the AIXM XML Schema. This schema is represented in the AIXM abstract UML package and applied to every AIXM Feature.

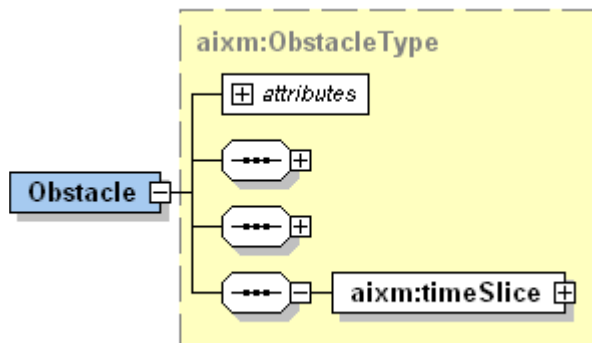


3.1.2 Mapping Features and Objects

For each AIXM Feature the following XML schema entities are created:

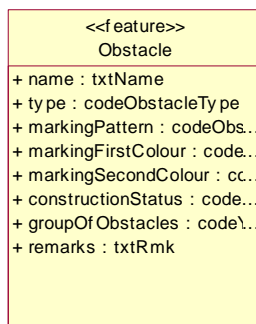
- `_FeatureExtension`
- `FeatureArrayPropertyType`
- `FeatureArray`
- `FeatureArrayType`
- `FeaturePropertyType`
- `FeaturePropertyGroup`
- `FeatureTimeSliceType`
- `FeatureTimeSlice.`
- `FeatureTimeSlicePropertyType`
- `FeatureType`
- `Feature`

To understand the schema entities listed above we need an understanding of the GML object-property model. The object/property model is encoded in GML by declaring a type and then assigning properties (attributes and relationships) to that type. The type defines the object.



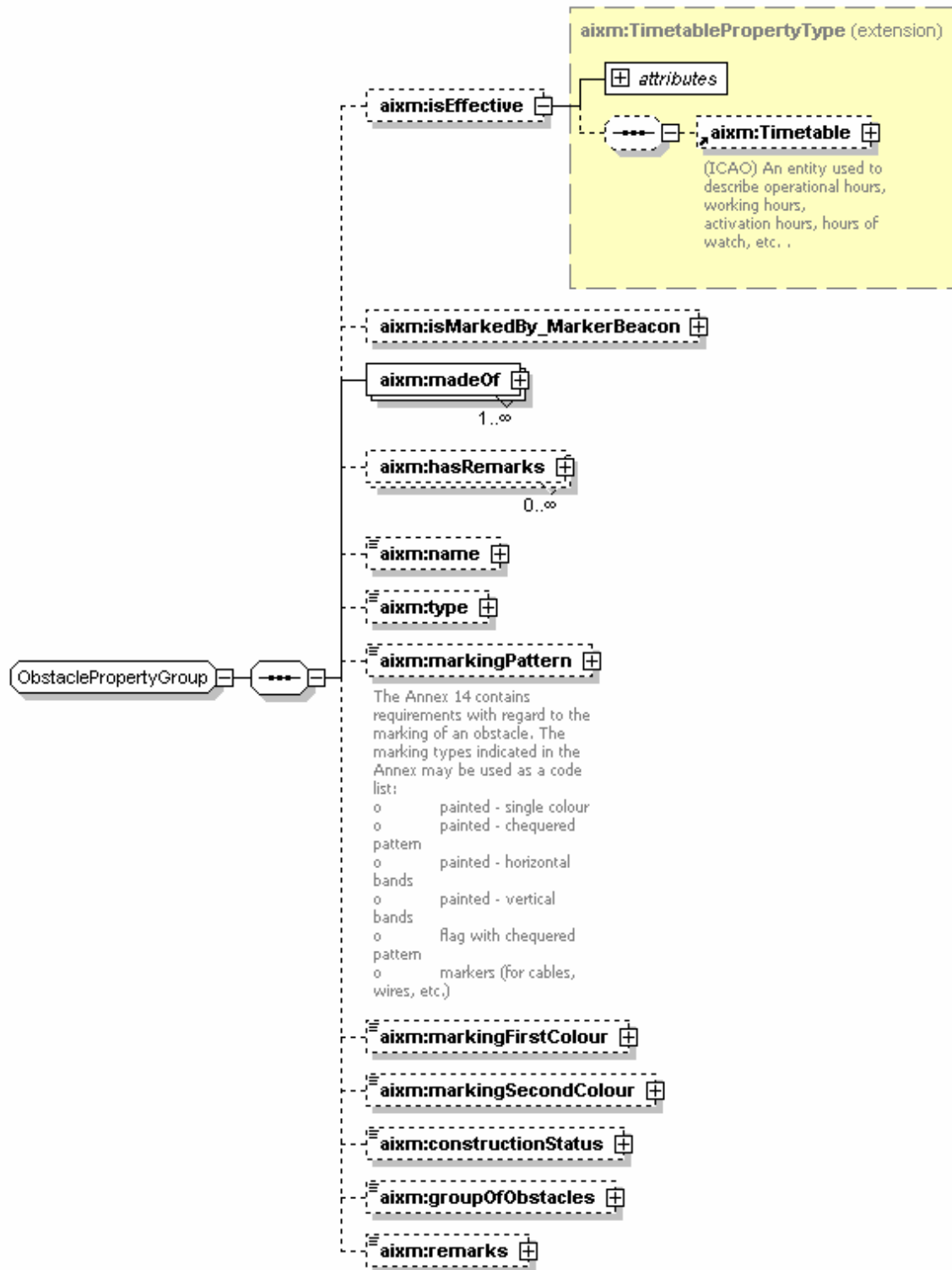
We will work backwards from the basic properties using the Obstacle feature to map UML to XSD.

These are described for the Obstacle feature:



3.1.1.1 ObstaclePropertyGroup

An XSD group is generated for each feature containing all the properties (attributes and relationships) of the feature. Below is an example of the ObstaclePropertyGroup. Note how the relationship 'isEffective' shown in section 2.2.3.2.1 Relationship to Objects is mapped below.



Generated by XmlSpy

www.altova.com

```

<group name="ObstaclePropertyGroup">
  <!-- START: sub makeGroupAttributes(cl as class) -->
  <sequence>
    <!-- START: sub EnumAssociations(cl as class) -->
    <!-- Association: <<>>isEffective -->
    <element name="isEffective" minOccurs="0">
      <complexType>
        <complexContent>
          <extension base="aixm:TimetablePropertyType"/>
        </complexContent>
      </complexType>
    </element>
    <!-- Association: <<>>isMarkedBy -->
    <element name="isMarkedBy_MarkerBeacon" nillable="true" minOccurs="0">
      <complexType>
        <complexContent>
          <extension base="aixm:MarkerBeaconPropertyType"/>
        </complexContent>
      </complexType>
    </element>
    <!-- Association: <<>>madeOf -->
    <element name="madeOf" maxOccurs="unbounded">
      <complexType>
        <complexContent>
          <extension base="aixm:ObstaclePartPropertyType"/>
        </complexContent>
      </complexType>
    </element>
    <!-- Association: <<>>hasRemarks -->
    <element name="hasRemarks" minOccurs="0" maxOccurs="unbounded">
      <complexType>
        <complexContent>
          <extension base="aixm:NotePropertyType"/>
        </complexContent>
      </complexType>
    </element>
    <element name="name" nillable="true" minOccurs="0">
      <complexType>
        <simpleContent>
          <extension base="aixm:txtName">
            <attributeGroup ref="aixm:NilReasonGroup"/>
          </extension>
        </simpleContent>
      </complexType>
    </element>
    <element name="type" nillable="true" minOccurs="0">
      <complexType>
        <simpleContent>
          <extension base="aixm:codeObstacleType">
            <attributeGroup ref="aixm:NilReasonGroup"/>
          </extension>
        </simpleContent>
      </complexType>
    </element>
    <element name="markingPattern" nillable="true" minOccurs="0">
      <annotation>
        <documentation>The Annex 14 contains requirements with regard to
the marking of an obstacle. The marking types indicated in the Annex may be used as a code list:
o painted - single colour
o painted - chequered pattern
o painted - horizontal bands
o painted - vertical bands
o flag with chequered pattern
o markers (for cables, wires, etc.)</documentation>
      </annotation>
    </complexType>
    <simpleContent>
      <extension base="aixm:codeObstacleMarkingPattern">
        <attributeGroup ref="aixm:NilReasonGroup"/>
      </extension>
    </simpleContent>
  </complexType>
</element>
<element name="markingFirstColour" nillable="true" minOccurs="0">
  <complexType>
    <simpleContent>

```

```

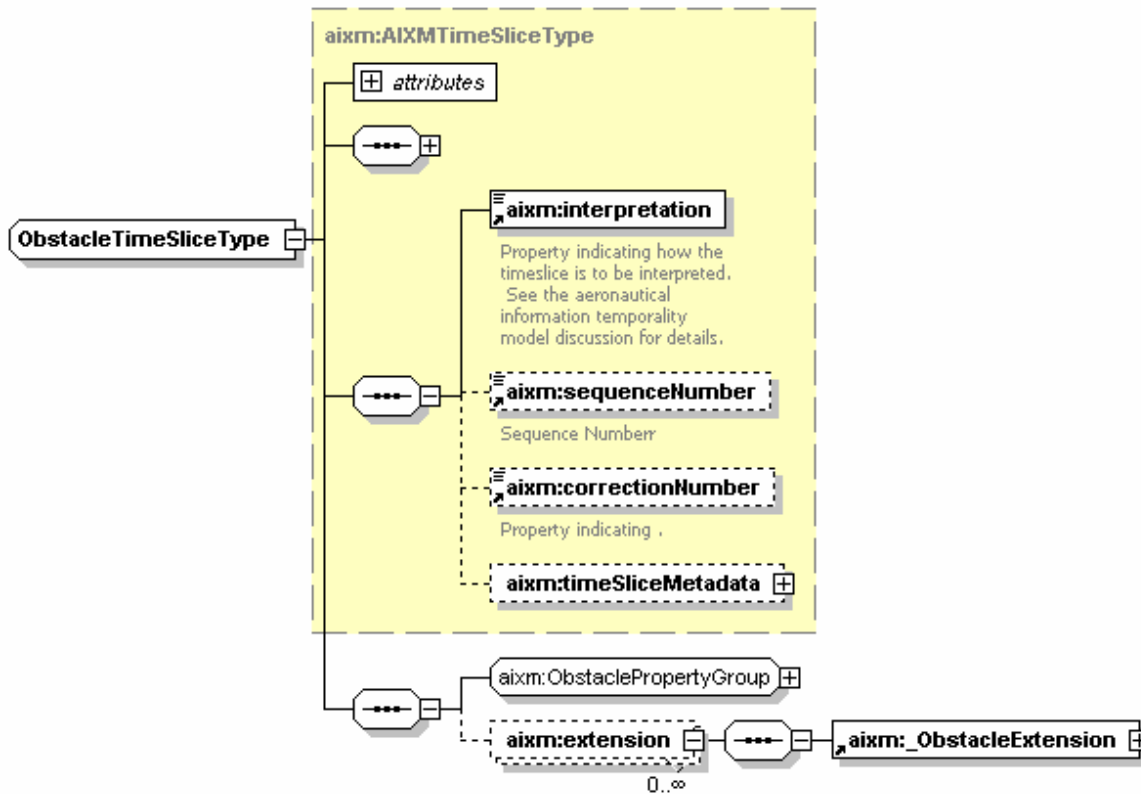
        <extension base="aixm:codeColour">
          <attributeGroup ref="aixm:nilReasonGroup"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
  <element name="markingSecondColour" nillable="true" minOccurs="0">
    <complexType>
      <simpleContent>
        <extension base="aixm:codeColour">
          <attributeGroup ref="aixm:nilReasonGroup"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
  <element name="constructionStatus" nillable="true" minOccurs="0">
    <complexType>
      <simpleContent>
        <extension base="aixm:codeStatusConstruction">
          <attributeGroup ref="aixm:nilReasonGroup"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
  <element name="groupOfObstacles" nillable="true" minOccurs="0">
    <complexType>
      <simpleContent>
        <extension base="aixm:codeYesNo">
          <attributeGroup ref="aixm:nilReasonGroup"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
  <element name="remarks" nillable="true" minOccurs="0">
    <complexType>
      <simpleContent>
        <extension base="aixm:txtRmk">
          <attributeGroup ref="aixm:nilReasonGroup"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
  <!-- START: sub processLinkClass(cl as class) -->
  <!-- end: sub processLinkClass(cl as class) -->
</sequence>
<!-- end: sub makeGroupAttributes(cl as class) -->
</group>

```

3.1.1.2 ObstacleTimeSliceType

The properties of a feature or the target of any feature relationship can change within the lifetime of the feature. This temporality can be expressed in GML by using dynamic features and feature collections. The `timeSlice` property of a dynamic feature contains one or more Feature Timeslices that capture the evolution of the feature over time. A `gml:TimeSlice` object contains the dynamic properties of the feature.

For each feature we create a `timeSlice` property that contains an array of feature `TimeSlice` objects. This example shows the `ObstacleTimeSlice` type encapsulating all of the `Obstacle` properties (`ObstaclePropertyGroup` created above) that change over time.



Generated by XmlSpy

www.altova.com

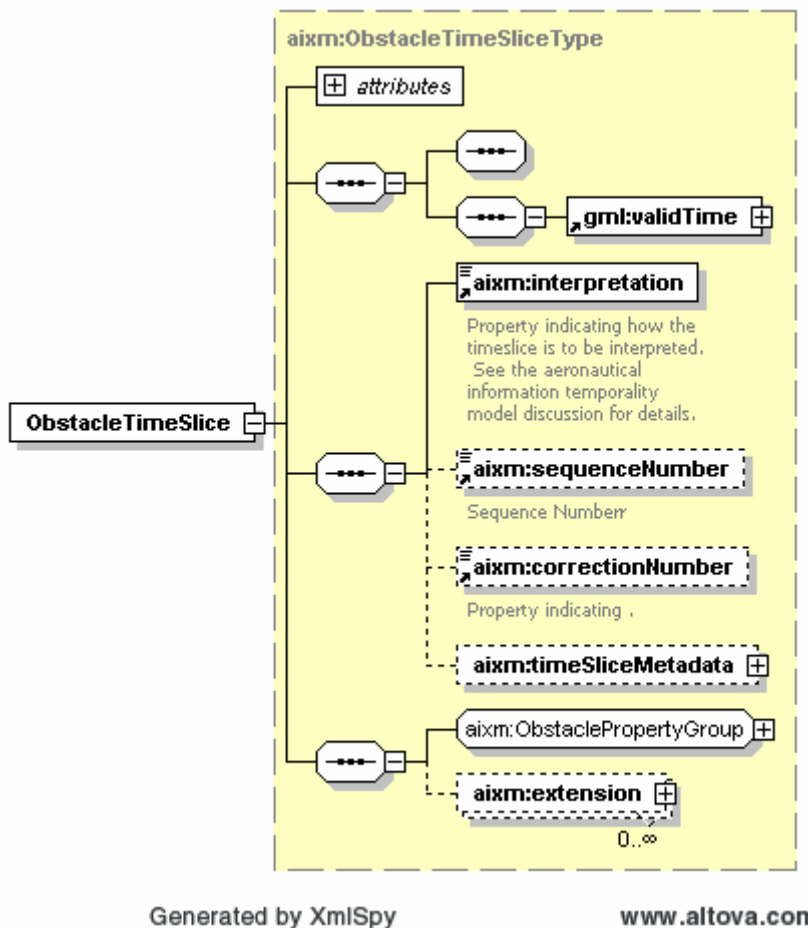
```

<complexType name="ObstacleTimeSliceType">
  <complexContent>
    <extension base="aixm:AIXMTimeSliceType">
      <sequence>
        <group ref="aixm:ObstaclePropertyGroup"/>
        <element name="extension" minOccurs="0">
          <complexType>
            <sequence maxOccurs="unbounded">
              <element ref="aixm:_ObstacleExtension"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </extension>
  </complexContent>
</complexType>

```

3.1.1.3 ObstacleTimeSlice

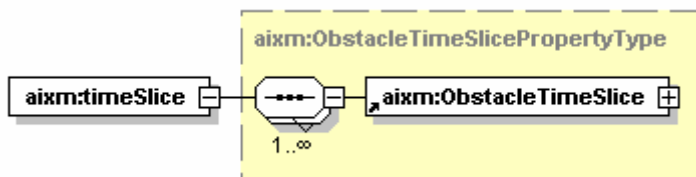
The TimeSlice object is of type FeatureTimeSliceType illustrated with the ObstacleTimeSlice diagram.



```
<element name="ObstacleTimeSlice" type="aixm:ObstacleTimeSliceType" abstract="false"
substitutionGroup="gml:_TimeSlice"/>
```

3.1.1.4 ObstacleTimeSlicePropertyType

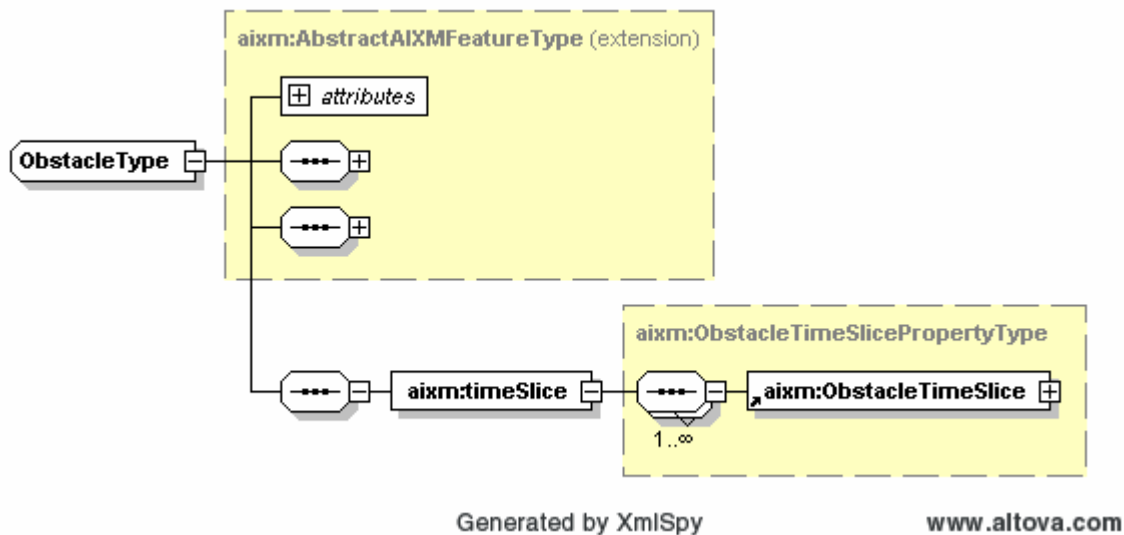
A GML property type containing an array of Obstacle TimeSlice objects.



```
<complexType name="ObstacleTimeSlicePropertyType">
  <sequence maxOccurs="unbounded">
    <element ref="aixm:ObstacleTimeSlice"/>
  </sequence>
</complexType>
```

3.1.1.5 ObstacleType

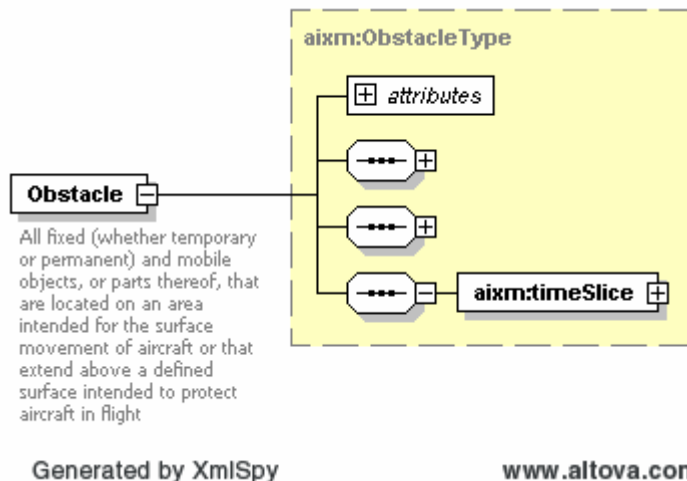
Continuing with the object-property model the Obstacle feature type is created extending the AbstractAIXMFeatureType with the ObstacleTimeSlice object created in 3.1.1.3 above.



```
<complexType name="ObstacleType">
  <complexContent>
    <extension base="aixm:AbstractAIXMFeatureType">
      <sequence>
        <element name="timeSlice"
          type="aixm:ObstacleTimeSlicePropertyType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

3.1.1.6 Obstacle

The obstacle feature is then defined by the ObstacleType.

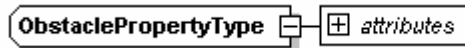


```
<element name="Obstacle" type="aixm:ObstacleType" abstract="false" substitutionGroup="aixm:_Feature">
  <annotation>
    <documentation>All fixed (whether temporary or permanent) and mobile
objects, or parts thereof, that are located on an area intended for the surface movement of aircraft or that extend above a
defined surface intended to protect aircraft in flight</documentation>
  </annotation>
</element>
```

3.1.1.7 ObstaclePropertyType

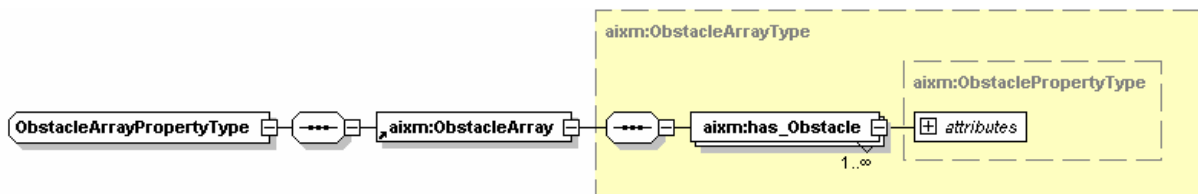
When a property of a feature is a relationship, the relationship must be associated to another feature or object. This is done through the creation of the *featurePropertyType*, in this case, the *ObstaclePropertyType*.

In AIXM, when the relationship or association points to another feature the feature is referenced using the *xlink:href* attribute as shown below. When the relationship points to an object, we include the object inside the parent. (Objects can not exist without the parent.) Since an *Obstacle* is a feature the *ObstaclePropertyType* is created with the attribute *xlink:href*.



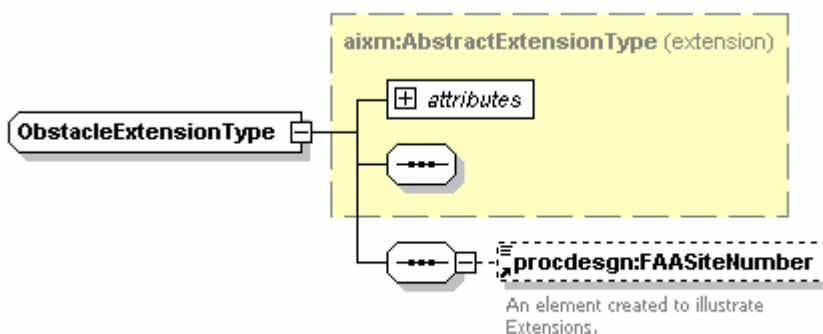
3.1.1.8 ObstacleArrayPropertyType, ObstacleArray and ObstacleArrayType

When the relationship has a multiplicity greater than one an array property type must be created. Following the object/property model discussed above the *ObstacleArrayType* containing the one-to-many relationship is created. The *ObstacleArrayType* is then attached to the *ObstacleArray* object.



3.1.1.9 ObstacleExtension

All Features and objects can be extended. A relationship is created with an abstract XML element that acts as the head for all extensions. Below is an example of the extension for the *Obstacle* feature. We created the *ObstacleExtensionType* with the new element with the base *AbstractExtensionType* as shown below.

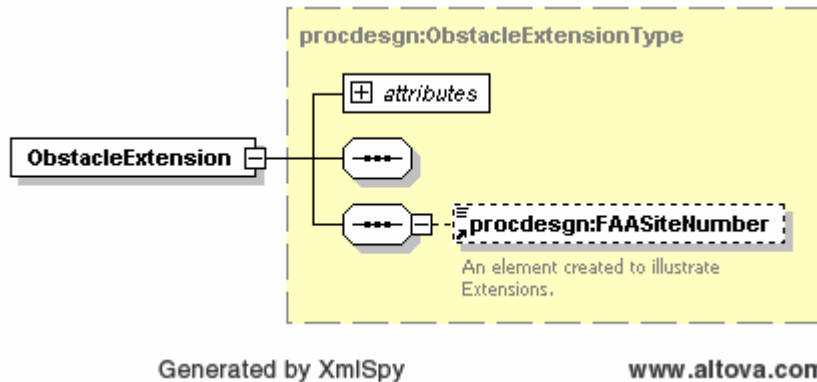


```

Error!
Generated by XmlSpy www.altova.com

<xs:complexType name="ObstacleExtensionType">
  <xs:complexContent>
    <xs:extension base="aixm:AbstractExtensionType">
      <xs:sequence>
        <xs:element ref="procdesgn:FAASiteNumber" minOccurs="0">
          <xs:annotation>
            <xs:documentation>An
            element created to illustrate Extensions.</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
    
```

Then we created the `ObstacleExtension` element assigning the `ObstacleExtensionType` created above as the base.



```
<xs:element name="ObstacleExtension" type="procdesgn:ObstacleExtensionType"
substitutionGroup="aixm:_ObstacleExtension"/>
```

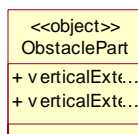
3.1.2 Objects

We encode AIXM objects as GML objects. For the most part, the XML schema entities are created the same as the feature following the object-property model. The exception is AIXM objects do not exist outside of a feature therefore they are part of the feature timeslice. Timeslice schemas are not created for objects.

For each AIXM Object the following XML schema entities are created:

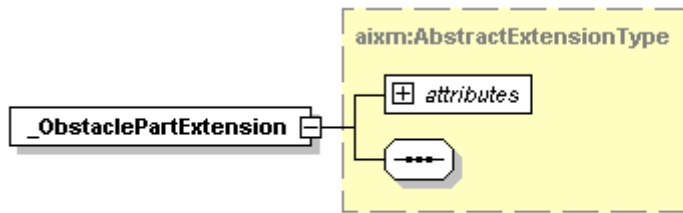
- `_ObjectExtension`
- `ObjectPropertyGroup`
- `ObjectType`
- `Object`
- `ObjectPropertyType`
- `ObjectArrayPropertyType`
- `ObjectArray`
- `ObjectArrayType`
-

These are described for the `ObstaclePart` object:



3.1.2.1 ObstaclePartExtension

An abstract XML element that acts as the head for all extension to the `ObstaclePart` object. Object extensions are defined the same as Feature extensions.



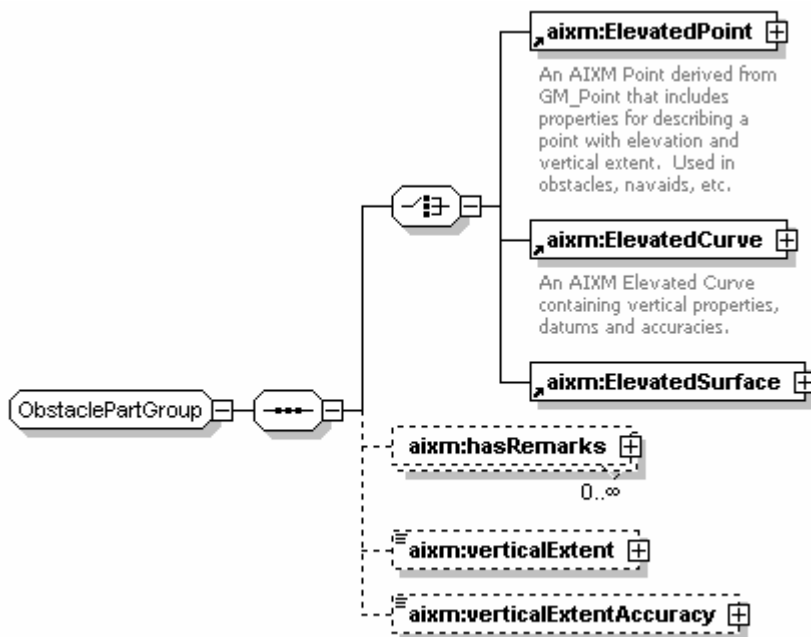
Generated by XmlSpy

www.altova.com

```
<element name="_ObstaclePartExtension" type="aixm:AbstractExtensionType" abstract="true"
substitutionGroup="aixm:_Extension"/>
```

3.1.2.2 ObstaclePartPropertyGroup

An XSD group containing the properties of the ObstaclePart object is created as we did for Features.



Generated by XmlSpy

www.altova.com

```
<group name="ObstaclePartGroup">
  <sequence>
    <choice>
      <element ref="aixm:ElevatedPoint"/>
      <element ref="aixm:ElevatedCurve"/>
      <element ref="aixm:ElevatedSurface"/>
    </choice>
    <element name="hasRemarks" minOccurs="0" maxOccurs="unbounded">
      <complexType>
        <complexContent>
          <extension
            base="aixm:NotePropertyType"/>
        </complexContent>
      </complexType>
    </element>
    <element name="verticalExtent" nillable="true" minOccurs="0">
```

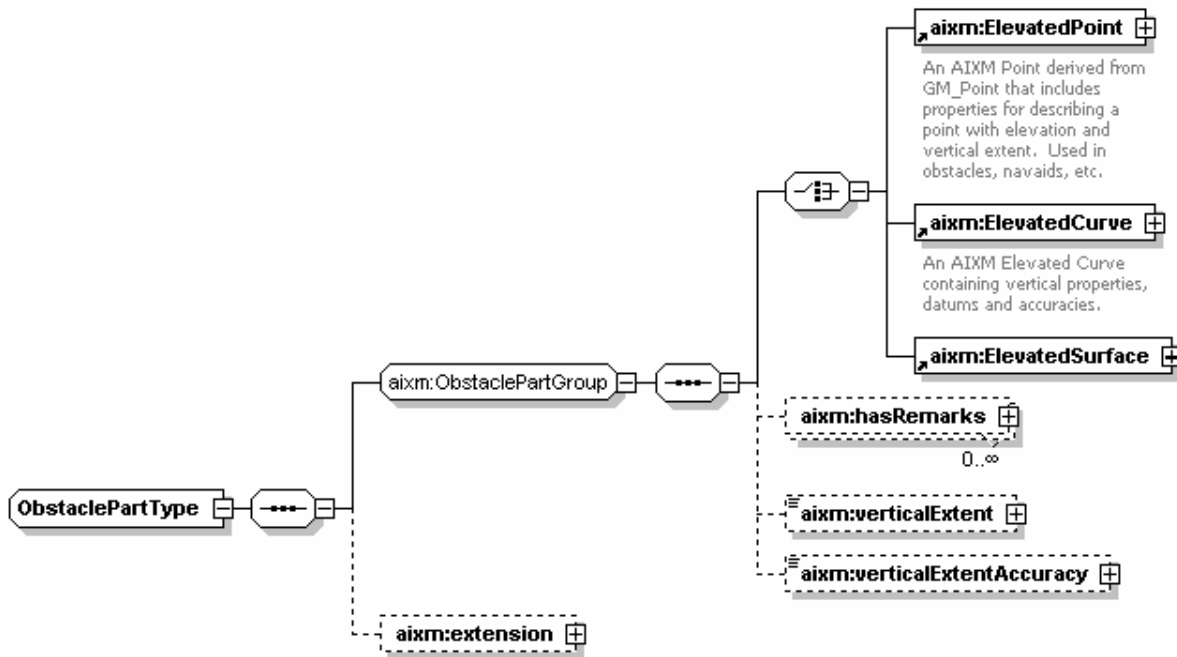
```

<complexType>
  <complexContent>
    <extension
      base="aixm:valDistVer">
      <attributeGroup ref="aixm:NilReasonGroup"/>
    </extension>
  </complexContent>
</complexType>
</element>
<element name="verticalExtentAccuracy" nillable="true" minOccurs="0">
  <complexType>
    <complexContent>
      <extension
        base="aixm:valDistVer">
        <attributeGroup ref="aixm:NilReasonGroup"/>
      </extension>
    </complexContent>
  </complexType>
</element>
</sequence>
</group>

```

3.1.2.3 ObstaclePartType

The ObstaclePart Object type definition.



Generated by XmlSpy

www.altova.com

```

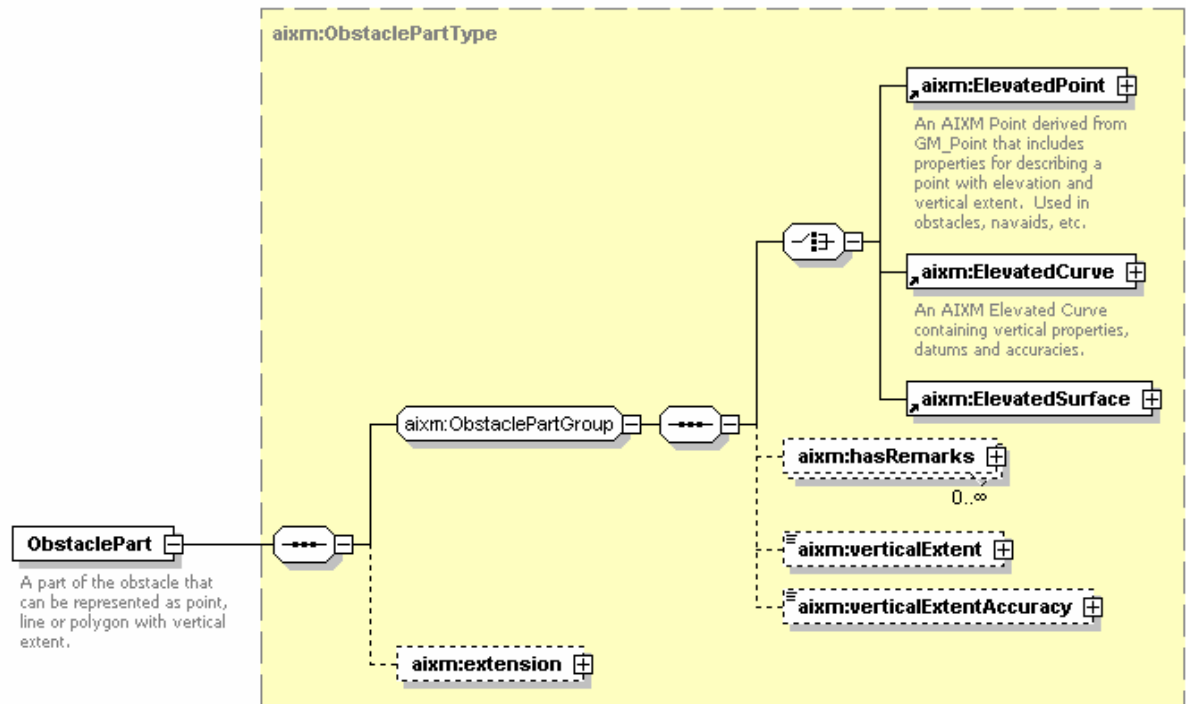
<complexType name="ObstaclePartType">
  <sequence>
    <group ref="aixm:ObstaclePartGroup"/>
    <element name="extension" minOccurs="0">
      <complexType>
        <sequence maxOccurs="unbounded">
          <element
            ref="aixm:_ObstaclePartExtension"/>
        </sequence>
      </complexType>
    </element>
  </sequence>
</complexType>

```

</complexType>

3.1.2.4 ObstaclePart

The ObstaclePart object defined as an XSD element.



Generated by XmlSpy

www.altova.com

```
<element name="ObstaclePart" type="aixm:ObstaclePartType" abstract="false">
```

```
  <annotation>
```

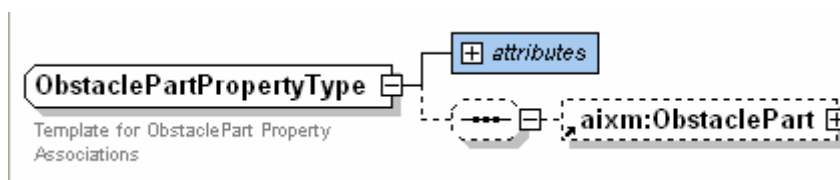
```
    <documentation>A part of the obstacle that can be represented as point, line or polygon with vertical extent.</documentation>
```

```
  </annotation>
```

```
</element>
```

3.1.2.5 ObstaclePartPropertyType

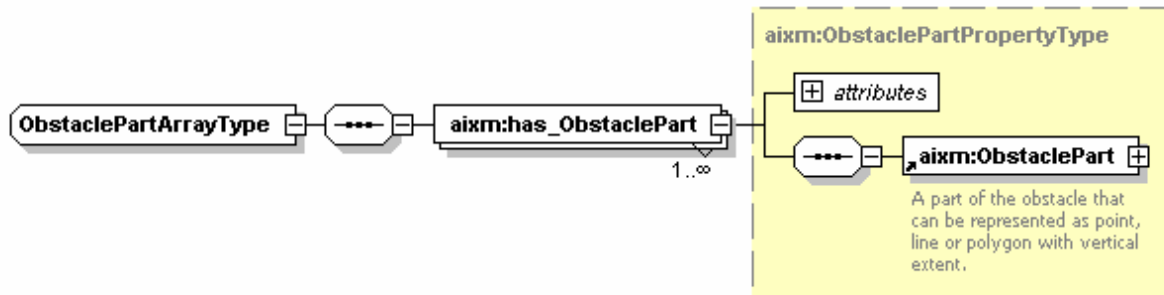
An XSD complex type representing a GML property type. Used when the ObstaclePart is the child of a relationship with multiplicity 0..1. Here the ObstaclePart object is included instead of referenced because objects do not exist without the parent.



```
<complexType name="ObstaclePartPropertyType">
  <annotation>
    <documentation>Template for ObstaclePart Property
Associations</documentation>
  </annotation>
  <sequence minOccurs="0">
    <element ref="aixm:ObstaclePart" minOccurs="0"/>
  </sequence>
  <attributeGroup ref="aixm:OwnershipAttributeGroup"/>
</complexType>
```

3.1.2.6 ObstaclePartArrayType

An XSD complex type representing the GML property array type. Used when the ObstaclePart is the child of a relationship with multiplicity 0..*.



Generated by XmlSpy

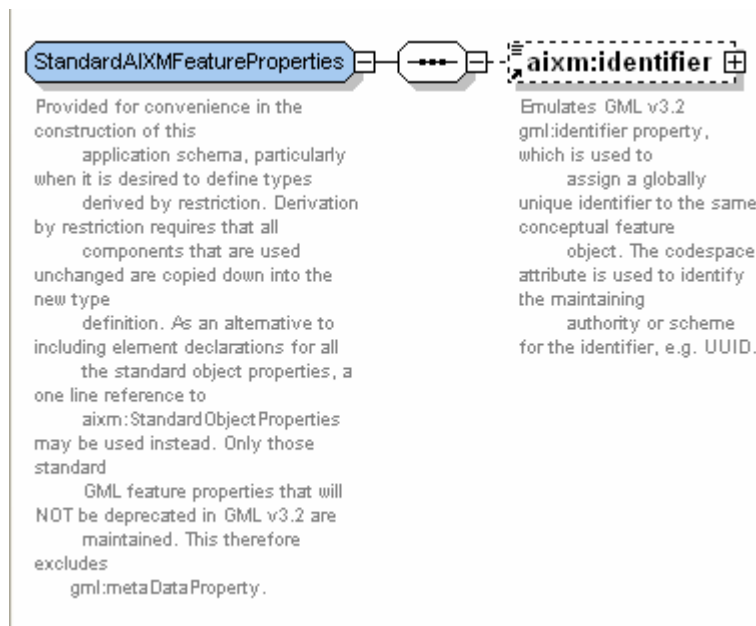
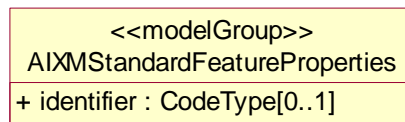
www.altova.com

```
<complexType name="ObstaclePartArrayType">
  <sequence>
    <element name="has_ObstaclePart" type="aixm:ObstaclePartPropertyType"
      maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

3.1.3 modelGroups

We encode AIXM modelGroups as XSD groups containing the common properties.

For example the AIXMStandardFeatureProperties modelGroup:



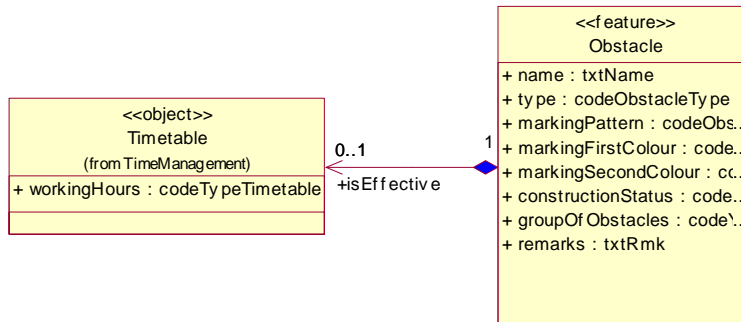
```
<group name="StandardAIXMFeatureProperties">
  <sequence>
    <element ref="aixm:identifier" minOccurs="0"/>
  </sequence>
```

```
</group>
```

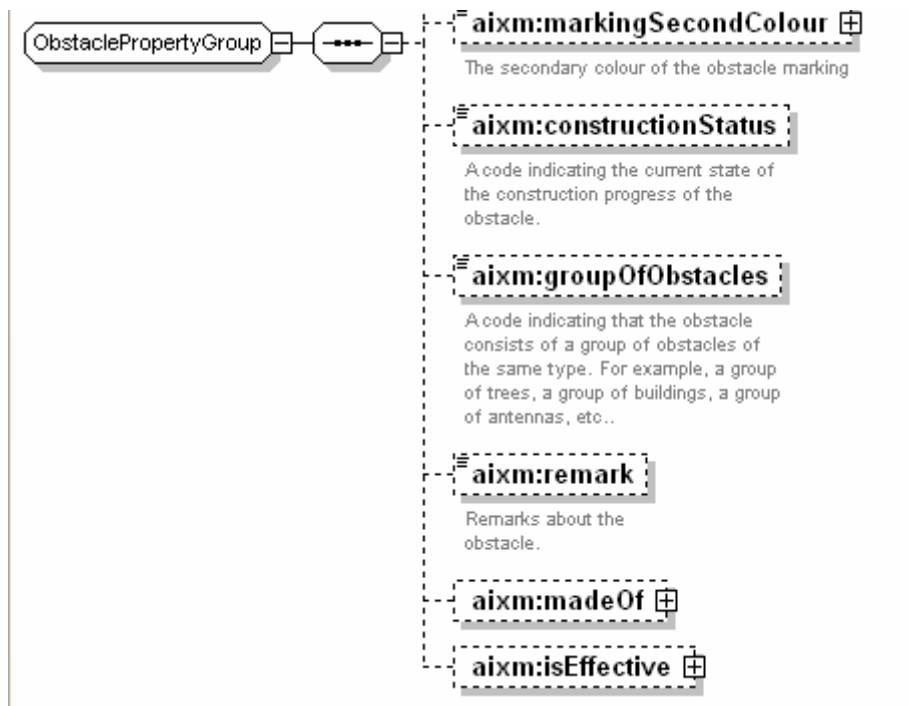
3.1.4 Relationships to Objects

We encode relationships by creating an XML element with the same name as the rolename on the UML model. Depending on the multiplicity the element is either of type *ObjectPropertyType* or *ObjectPropertyArrayType*.

In this example:

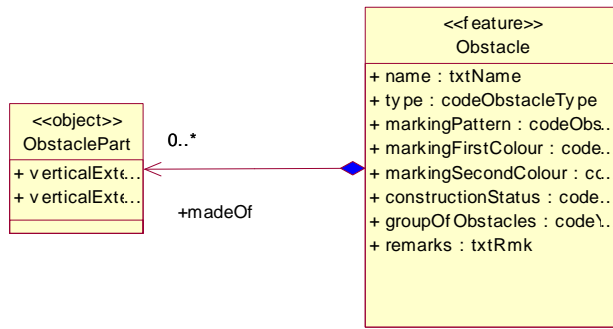


The TimeTable Object is a property of the Obstacle. The **isEffective** property of the Obstacle is defined as being of type TimeTablePropertyType.

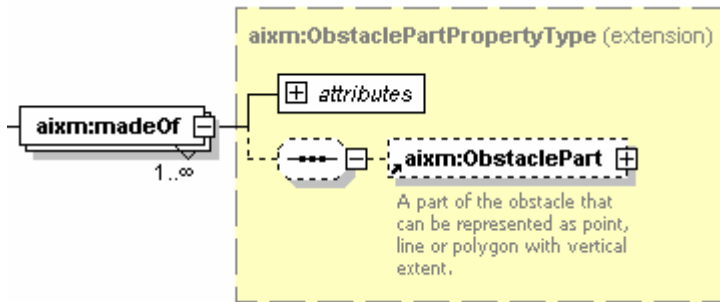


```
<element name="isEffective" type="aixm:TimeTablePropertyType"
minOccurs="0"/>
```

In this second example:



The Obstacle feature has a **madeOf** property related to an array of ObstaclePart Objects.

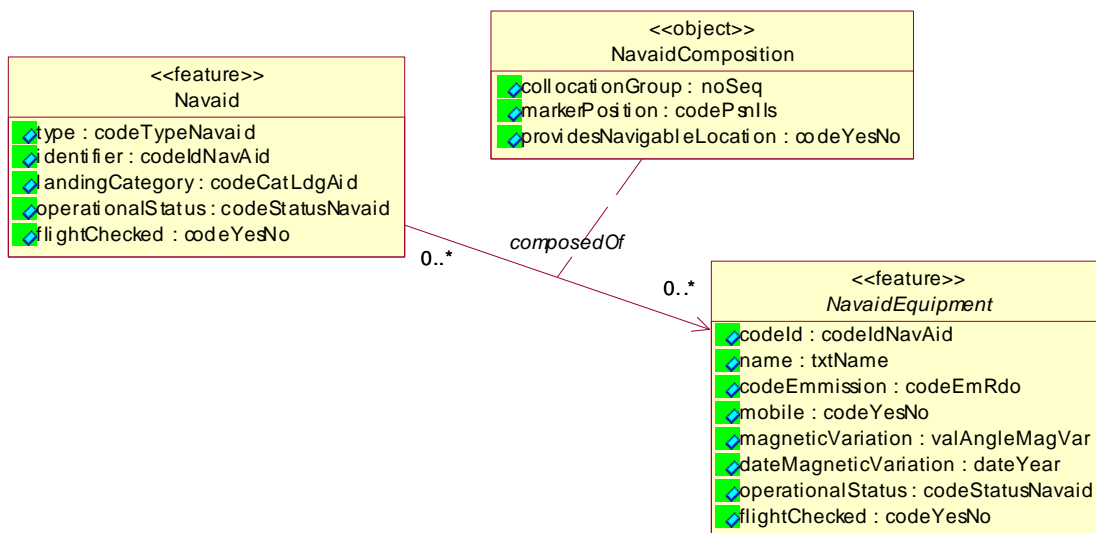


```

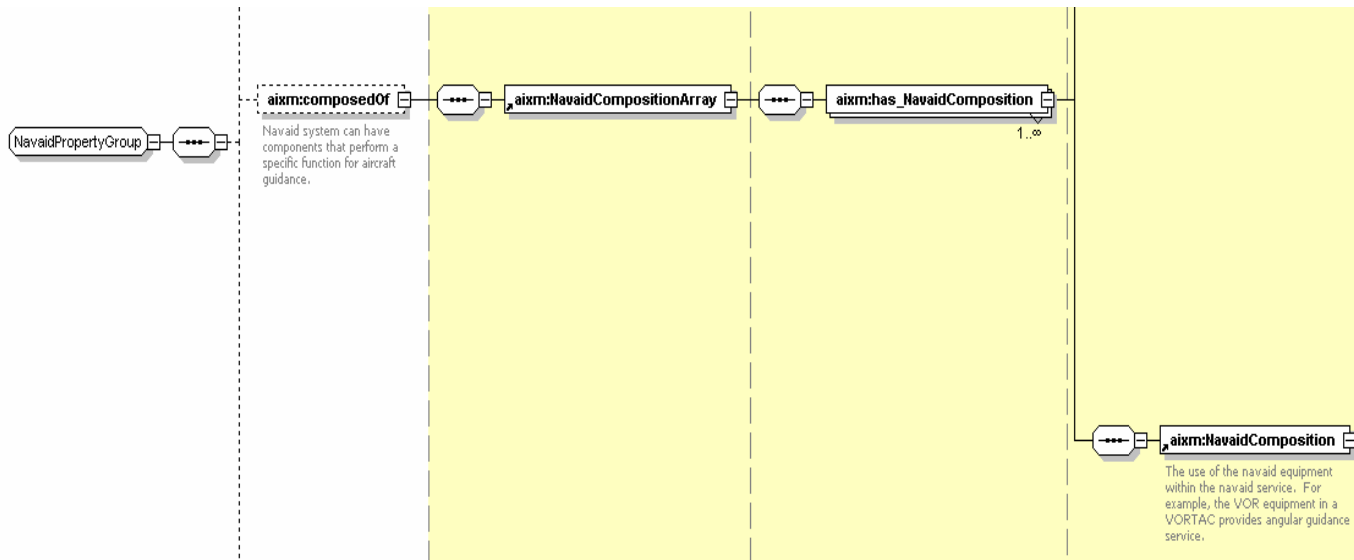
<element name="madeOf" maxOccurs="unbounded">
  <complexType>
    <complexContent>
      <extension base="aixm:ObstaclePartPropertyType"/>
    </complexContent>
  </complexType>
</element>

```

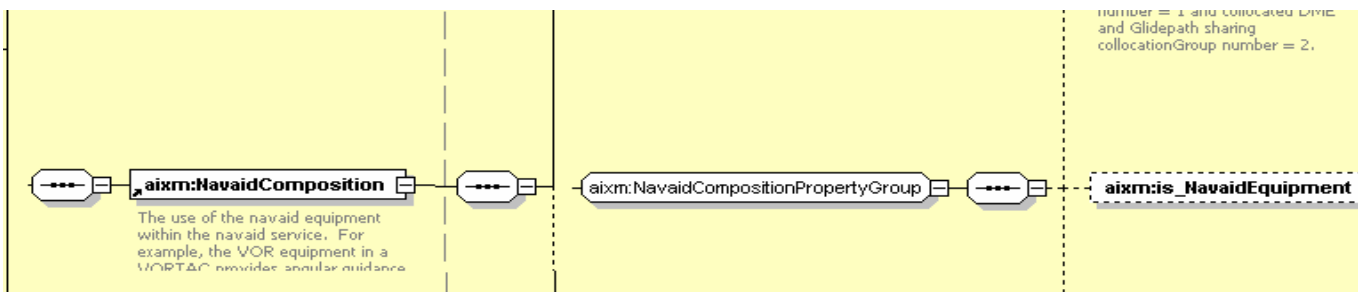
In the third example:



In the UML above, the Navaid feature has a relationship to the NavaidEquipment feature. This relationship contains properties defined in the NavaidComposition object. Mapping this in XSD we create a 'composedOf' property in the Navaid feature as shown below. (The direction of the arrow is important. If the direction would have been to the Navaid, the property would have been created in NavaidEquipment feature.)



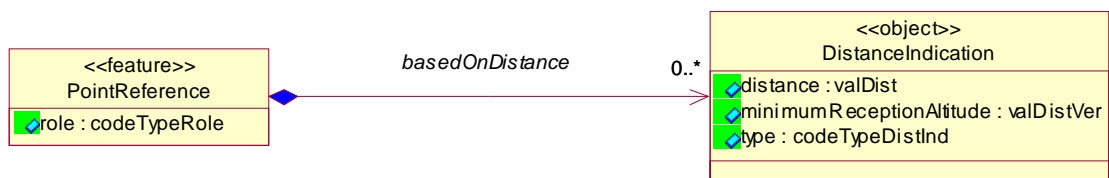
Within NavaidComposition object we create the 'is_NavidEquipment' property.



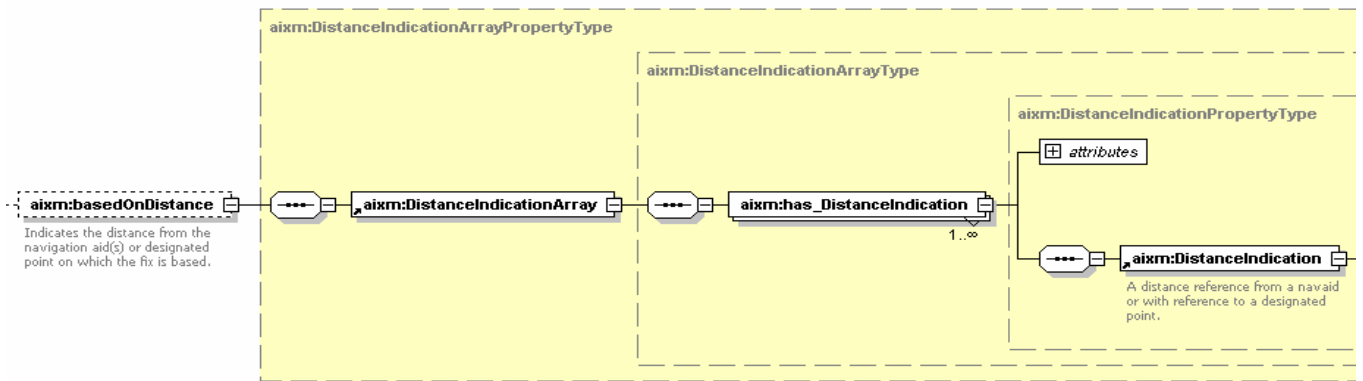
3.1.5 Relationships to Features

In AIXM, Relationships to features are described remotely using `xlink:href`. We use the UML rolename for the XML element name and the XML element is either of type `FeaturePropertyType` or `FeaturePropertyArrayType` depending on the relationship multiplicity.

A `PointLocation` feature may be based on many `DistanceIndication` features.



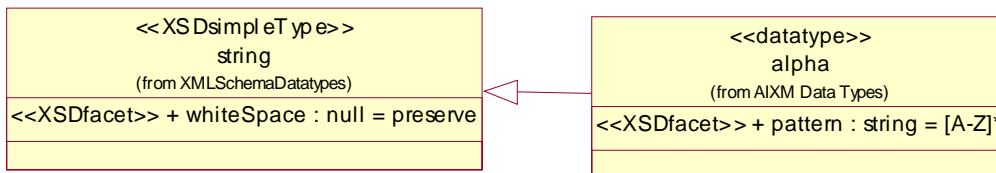
The `baseOnDistance` XML element is defined by an `ArrayPropertyType`.



3.2 Data Types

The default datatypes in a class map directly to the built-in datatypes defined by the XML schema specification. The default mapping of a class to a schema produces a `complexType`. The default datatypes are `string`, `float`, `double`, etc, which are considered `simpleTypes`. The user-defined datatypes are generated by the schema generator by including a stereotype on the UML class, for example, `<<datatype>>`. The datatypes are stereotyped as follows:

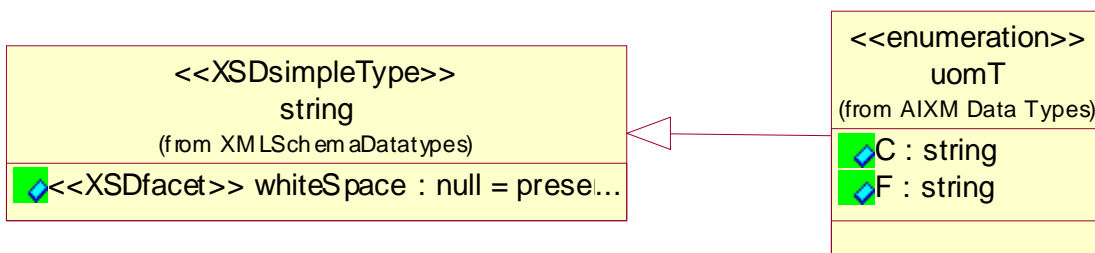
- `<<datatype>>` - encapsulates data



The equivalent XML schema snippet follows:

```
<simpleType name="alpha">
  <restriction base="xsd:string">
    <pattern value="[A-Z]*/>
  </restriction>
</simpleType>
```

- `<<enumeration>>` - a constant list of values
-



The equivalent XML schema snippet follows:

```
<xsd:simpleType name="uomT" >
```

```

    <xsd:annotation>
      <xsd:documentation>A unit of measurement for
temperature. </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="C">
        <xsd:annotation>

          <xsd:documentation>Degrees Celsius.</xsd:documentation>
        </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration value="F">
        <xsd:annotation>
          <xsd:documentation>Degrees
Fahrenheit.</xsd:documentation>
        </xsd:annotation>
      </xsd:enumeration>
    </xsd:restriction>
  </xsd:simpleType>

```

- `<<codelist>>` - similar to an enumeration used to indicate a list of possible values that can be expanded [1].

<code><<codelist>></code> codeNilReason
+ inapplicable : string
+ missing : string
+ template : string
+ unknown : string
+ with held : string

The equivalent XML schema snippet follows:

```

    <simpleType name="codeNilReason">
      <union memberTypes="aixm:codeNilReason_base
xsd:string"/>
    </simpleType>
    <simpleType name="codeNilReason_base">
      <annotation>
        <documentation/>
      </annotation>
      <restriction base="xsd:string"/>
    </simpleType>

```

The schema generator recognizes the stereotype and generates this class as a `simpleType` by combining two or more member classes using `xsd:union`. The `xsd:string` as a member specifies that a value can be one from the pre-specified list or a custom value.

```

<simpleType name="codeNilReason">
  <union memberTypes="aixm:codeNilReason_base xsd:string"/>
</simpleType>

```

```

<simpleType name="codeNilReason_base">
  <annotation>
    <documentation/>
  </annotation>
  <restriction base="xsd:string">
    <enumeration value="inapplicable">
      <annotation>
        <documentation/>

```

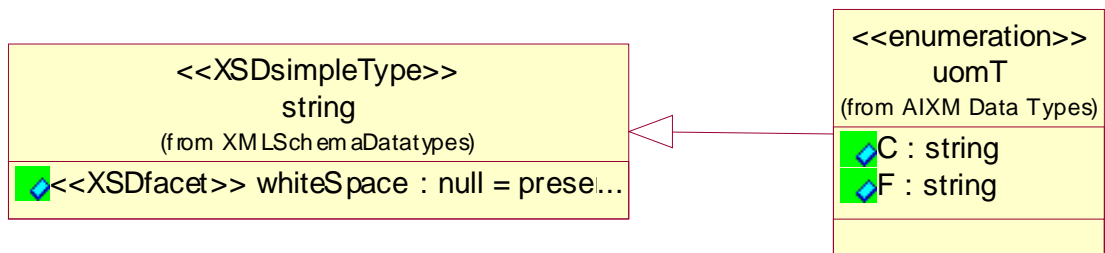
```

        </annotation>
    </enumeration>
    <enumeration value="missing">
        <annotation>
            <documentation/>
        </annotation>
    </enumeration>
    <enumeration value="template">
        <annotation>
            <documentation/>
        </annotation>
    </enumeration>
    <enumeration value="unknown">
        <annotation>
            <documentation/>
        </annotation>
    </enumeration>
    <enumeration value="withheld">
        <annotation>
            <documentation/>
        </annotation>
    </enumeration>
</restriction>
</simpleType>

```

3.3 Unit of Measurement

A Unit of measurement (UOM) exist for every value attribute. This has been illustrated as a containment relationship drawn from the value class to the UOM class.



The equivalent xml schema snippet follows:

```

<simpleType name="valTBase">
    <annotation>
        <documentation/>
    </annotation>
    <restriction base="xsd:decimal">
        <pattern value="(\+|\-){0,1}\d{1,8}(\.\d{1,2}){0,1}" />
    </restriction>
</simpleType>
<complexType name="valT">
    <simpleContent>
        <annotation>
            <documentation/>
        </annotation>
        <extension base="aixm:valTBase">
            <attribute name="uom" type="aixm:uomT" use="required" />
        </extension>
    </simpleContent>
</complexType>

```

