AIXM

UML to XML Schema Mapping

Aeronautical Information Exchange Model (AIXM)

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
0.2	2007/07/31	Scott Wilson	Update for Release Candidate 2
0.3	2007/09/20	Eddy Porosnicu	Modified model for units of measurement
0.4	2008/01/15	Eddy Porosnicu	Modified rules for associations, from now on based on role names.
1.0	2008/03/15	Eddy Porosnicu	Editorial changes for first public version.
1.1	2010/02/04	Eddy Porosnicu Hubert Lepori	Updated for AIXM 5.1

CONTENTS

1 Scope

1.1 Introduction

The AIXM Conceptual Model is maintained as a UML class model. The AIXM exchange format is codified as a series of XML schemas. There is a direct link between the AIXM Conceptual Model and the AIXM XML Schema.

This document describes how the AIXM Conceptual Model is converted into the AIXM XML Schema. The conversion process is illustrated using a series of examples from the AIXM 5 XML schema.

1.2 References

1. Geographic Information – Spatial Schema. ISO 19107. First Edition, 2003-05-01
2. ISO 19136:2007 - Geographic information -- Geography Markup Language (GML)
3. UML 2.0 In a Nutshell. Dan Pilone. O'Reilly Media Inc. 2005.
4. AIXM Temporality Model, www.aixm.aero (see Downloads)

2 AIXM UML Modelling Conventions

2.1 Diagram types

Two types of diagrams are used in the model:

- Class diagrams – Used to represent the features, properties, relationships and inheritance between features;
- Package diagrams – Used to split the model into modules and identify dependencies among sets of classes.

2.2 Stereotypes

The classes are distinguished by their stereotypes. Stereotypes are used to further define and extend standard UML concepts. The main stereotype are `<<feature>>`, `<<object>>`, `<<choice>>`, `<<datatype>>` and `<<codelist>>`.

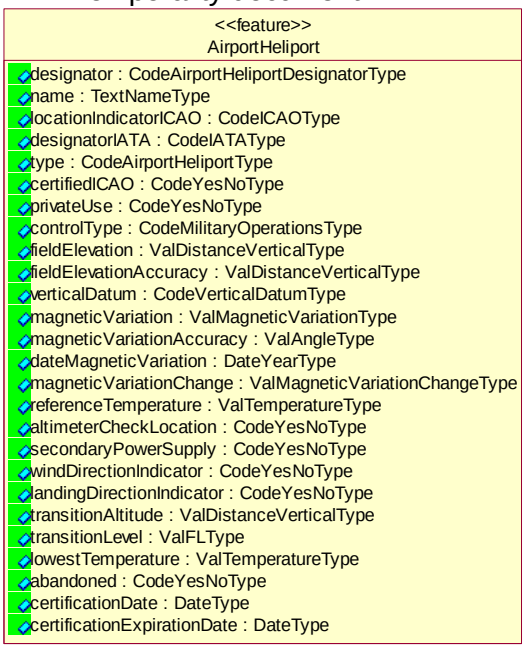
2.3 Abstract Classes

In addition, some classes are abstract. Abstract classes are designated by putting the class name in *italics*. An abstract class cannot be realised in an implementation such as an XML document. Instead, abstract classes are used as base classes in an inheritance hierarchy. For example, the AIXMFeature abstract class describes the basic properties of an AIXM Feature. Every specific AIXM Feature, such as Runway, inherits[1] from the abstract AIXMFeature class.

2.4 Features

Features describe real world entities and are fundamental in AIXM. AIXM features can be concrete and tangible, or abstract and conceptual and can change in time. Features are represented as classes with a stereotype `<<feature>>`. Examples include Runway and AirportHeliport.

AIXM features are dynamic features. Timeslice objects are used to describe the changes that affect the AIXM feature over time. Timeslice objects and temporality are discussed extensively in a separate AIXM Temporality document.



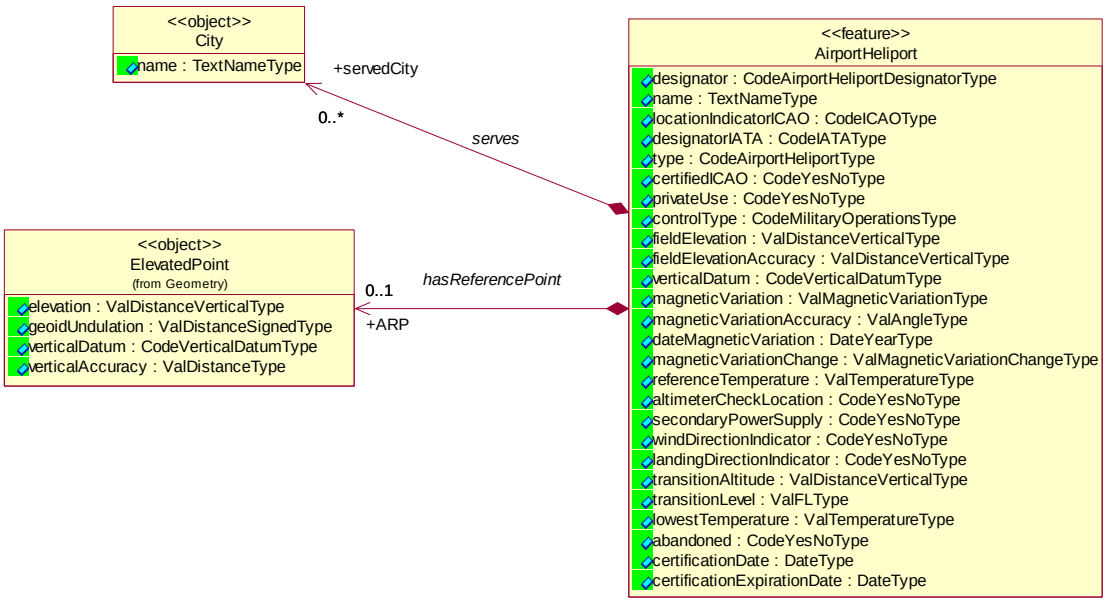
[1] Please see section 3.1 The Abstract Model, which explains why this inheritance is not visible in UML

2.5 Objects

Objects are abstractions of real world entities or, more frequently, of properties of these entities, which do not exist outside of a feature. An object is created for two reasons in AIXM:

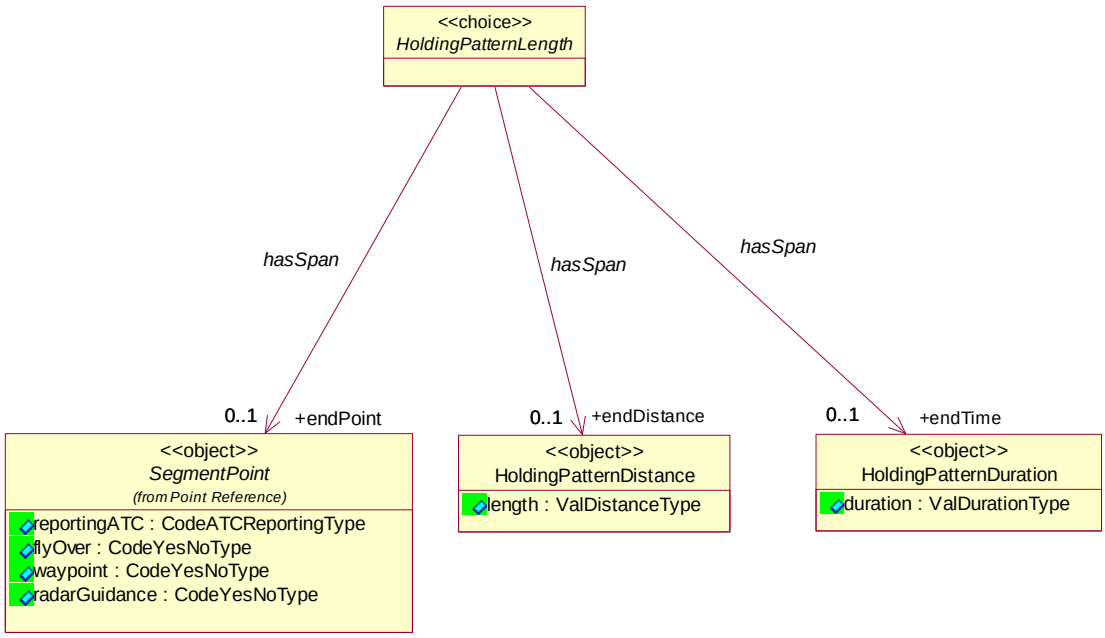
- When a property has a multiplicity greater than one (such as the city served by an AirportHeliport), or
- The object has its own attributes that are reused throughout the model, such as ElevatedPoint.

In both cases, the property is represented as an object with the proper UML composition relationship as shown below.



2.6 Choice

Some classes are marked as <<choice>>. These are used to model XOR relationships. For example, the length of a Holding Pattern can be expressed using a HoldingPatternDistance, a HoldingPatternDuration or a SegmentPoint defining the end of the outbound leg.



2.7 Properties

Properties are the attributes and relationships that characterise a feature or object. In the UML:

- Attributes are used to describe simple properties of a feature or object;
- Relationships are used to describe associations to features or objects. Whenever a property has a multiplicity greater than one, it is described using a UML relationship with cardinality.

2.7.1 Attributes

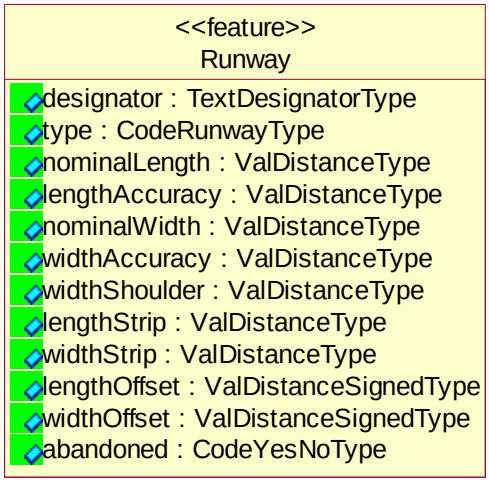
Simple properties of cardinality one are shown as attributes in the UML diagram.

An attribute has the following format:
Visibility / stereotype name : type multiplicity

For AIXM 5 the following values are used:

- Visibility – Public
- / – not used
- Stereotype – not used
- Name – name of the property
- Type – property type
- Multiplicity – usually not specified; for reasons related to the AIXM Temporality model, an implementation should assume that all properties are optional, [0..1]

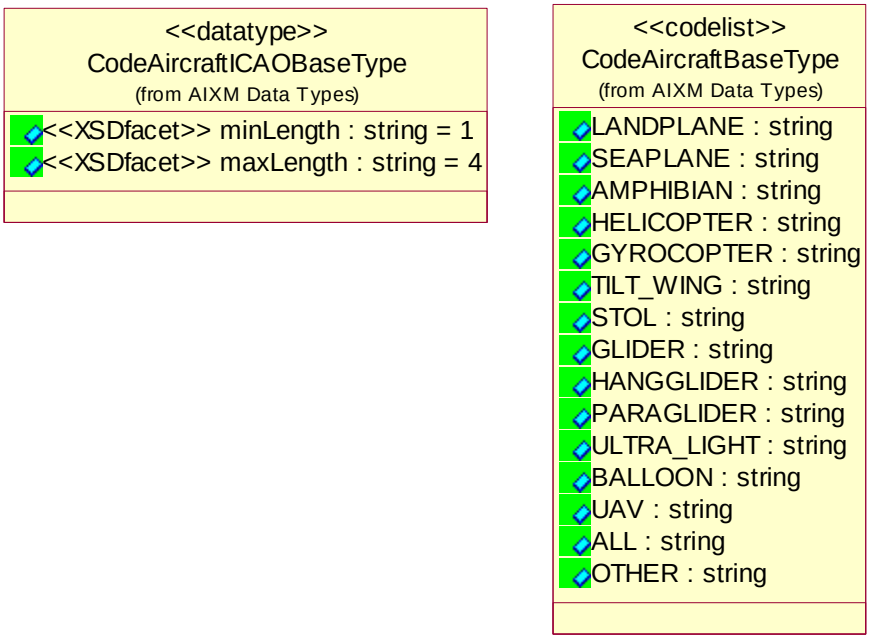
To illustrate, the Runway feature has several simple properties e.g. designator and type. These properties are assigned a datatype; for example, the designator attribute is of type TextDesignatorType.



2.7.1.1 DataTypes

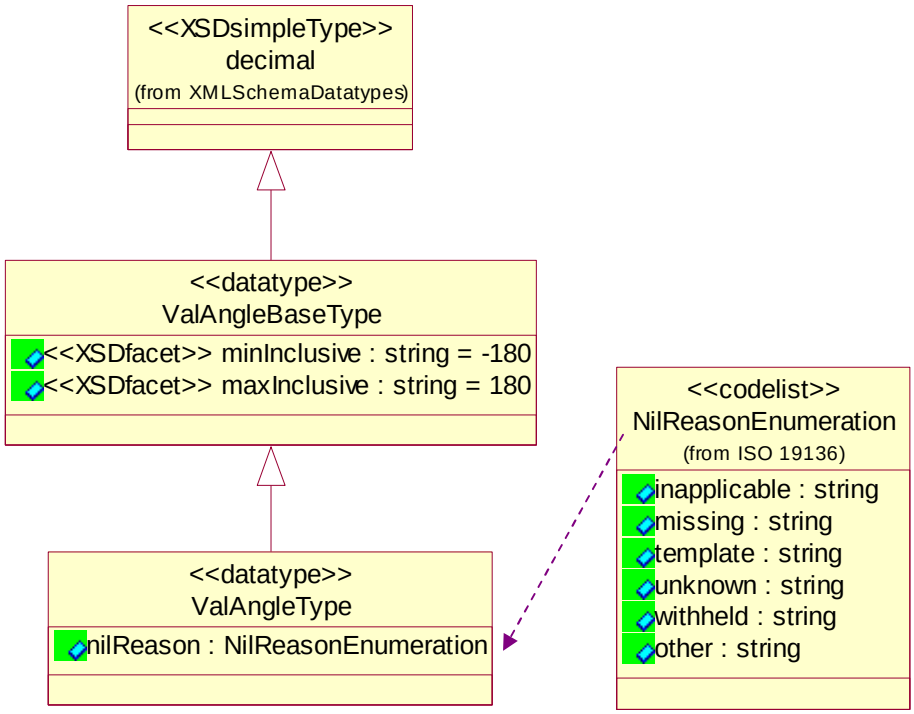
The UML model lists the datatypes that are used throughout the AIXM. These are given one of the two following stereotypes:

- <<datatype>> - This is basic data type that specifies a pattern to use.
- <<codelist>> - This is a data type which codes a predefined list of values. The <<codelist>> includes the value OTHER which can be expanded with some free text in uppercase ("OTHER:*MY_VALUE*") to support un-supported values.

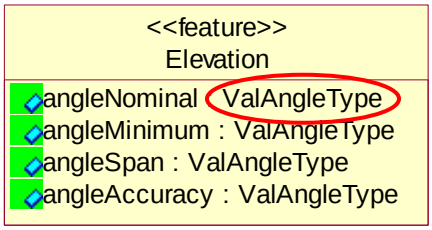


All the data types used to type AIXM simple properties define a nilReason, which is used to indicate the reason for a null value. This is realized in AIXM 5.1 by introducing

- A base type, which contains the core "business" information, such as a range of value for <<datatype>>, or the list of string values for <<codelist>>
- A derived data type, which explicitly declares the nilReason attribute, and which is used to type the corresponding AIXM simple properties.

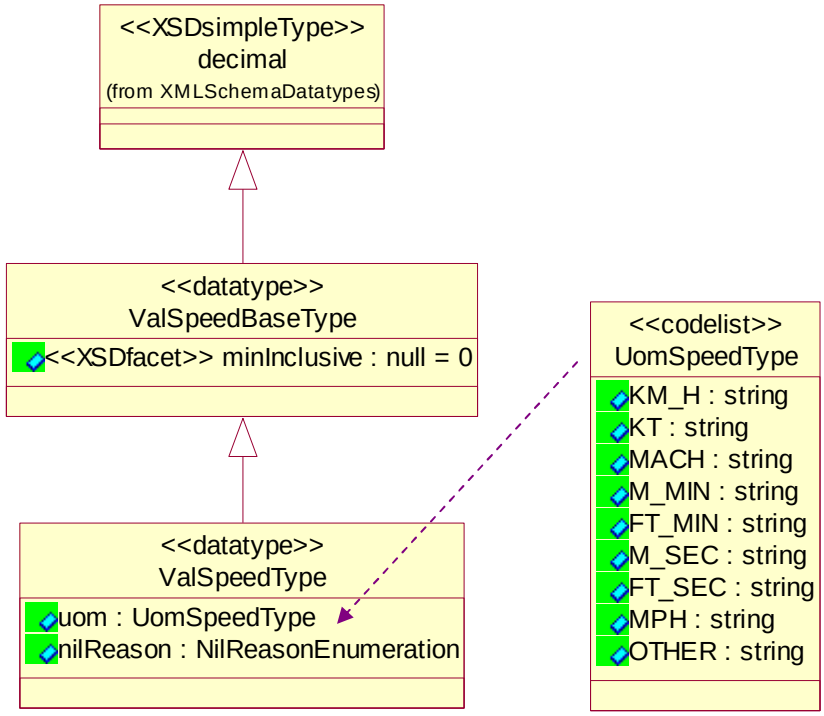


On the example above, the base type used to represent an angle is named ValAngle**BaseType**. It derives from decimal and defines the range of values allowed for an angle percentage ([-180;180]). The derived datatype ValAngle**Type** inherits from ValAngleBaseType and includes the nilReason, typed with NilReasonEnumeration. ValAngle**Type** is always used to type the percentages specified in AIXM features or AIXM objects.



A limited set of data types defined in the AIXM 5.1 UML model are not used to type directly AIXM simple properties but are basic classes from which several AIXM data types inherit. These data types are: AlphaType, AlphaNumericType, Character1, Character2, Character3. They do not require a nilReason attribute, and consequently, no corresponding BaseType types are defined in the AIXM UML model.

In addition, certain <<datatype>> might have an associated Unit Of Measurement. This is indicated in the model by the inclusion of a "uom" attribute at the same level as the nilReason attribure, i.e in the definition of the derived <<datatype>> class. The type of the uom attribute is typically a <<codelist>> class, as shown below:



Note that the <<codelist>> types representing Units of Measurement do not require a nilReason. As a consequence, no base type is created for uom.

2.7.2 Relationships

Whenever a property has a multiplicity greater than one, it can not be described in UML with an attribute. In that case, the property is described using a UML relationship which specifies the cardinality and which is always navigable in one and only one direction. The name of the complex property is given by the name of the role played by the targeted class.

2.7.2.1 Relationships to Objects

Relationships to objects are depicted by the standard UML composition (*aggregation by value)* association. Composition is a form of aggregation with strong ownership and coincident lifetime of the parts by the whole. The part is removed when the whole is removed.

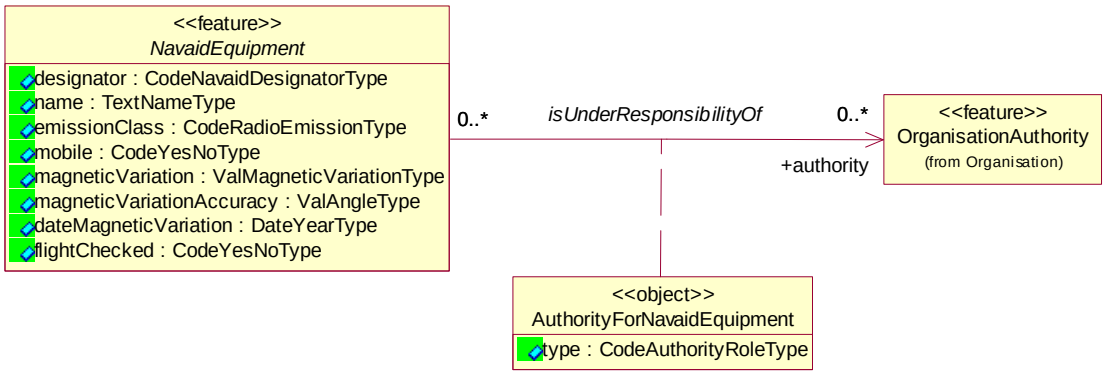
The example above shows that the <<feature>> Runway has a property named *theSurface.* This property is modelled in UML using a composition association between the <<feature>> Runway and an object representing the characteristics of a geometric surface.

2.7.2.2 Relationships to Features

Relationships to features are described with a standard UML association. All of the associations are navigable in only one direction. This shows that the two classes are related but only one class knows that the relationship exists. In the example below the Runway feature knows about the AirportHeliport but the AirportHeliport does not know about the Runway.

2.7.2.3 Association Classes

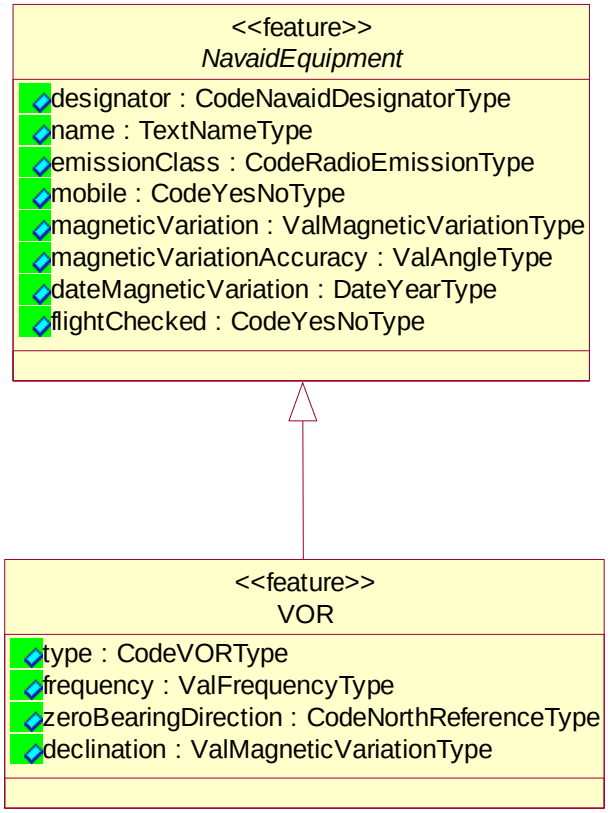
When information about a relationship is required, a UML association class is used. The association class is attached to the relationship with a dotted line.



2.8 Inheritance

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

In the example below the VOR is a kind of NavaidEquipment.



2.9 Naming

Feature, Object and Choice names are written in UpperCamelCase e.g. NavaidEquipment.

Simple property names (i.e. attributes) are written in lowerCamelCase e.g. widthShoulder. Relationship names are written in lowerCamelCase but as present tense verbs e.g. isSituatedAt. Relationship Role names are also written in lowerCamelCase and they are nouns that express the role played by the class in the association.

Datatype names are written in UpperCamelCase and end with ‘Type’ e.g. CodeAircraftType.

3 Other Aspects of the Model

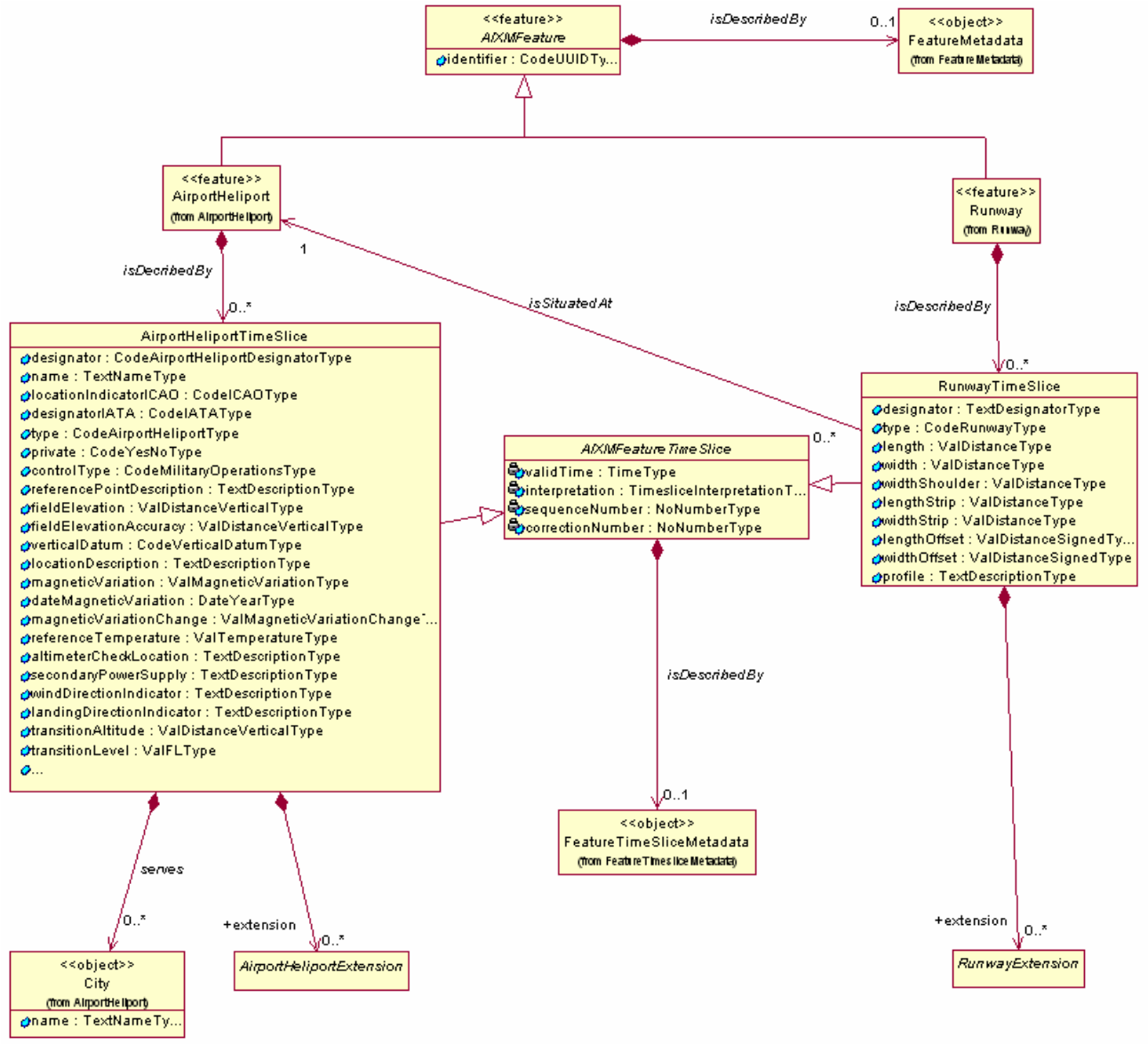
To simplify the UML model some convenience steps have been taken. Some elements are not shown on every diagram and some relationships are 'assumed'.

3.1 The Abstract Model

The model should contain a set of abstract AIXM classes that are used as the building blocks for the AIXM XML Schema. However, for simplicity, these relationships are not shown on any diagram and do not really exist in the UML. They are just assumed to exist, when converting from the UML model to the XML Schema of AIXM.

3.1.1 AIXMFeature and AIXMFeatureTimeSlice Class

The UML below shows how each and every <<feature>> inherits from the abstract AIXMFeature class. The concrete features are described by TimeSlices which are composed of properties. The TimeSlice inherits from the abstract AIXMFeatureTimeSlice class.



The diagram above is quite complex. If applied to the whole set of AIXM classes, it might undermine the readability of the UML diagrams. Therefore, the Design Team has decided to

provide a simplified UML model, without visible inheritance of all features from the abstract AIXMFeature and without visible SomeFeatureTimeSlice classes.

However, all of these relationships and classes must be mapped in the AIXM XML Schema.

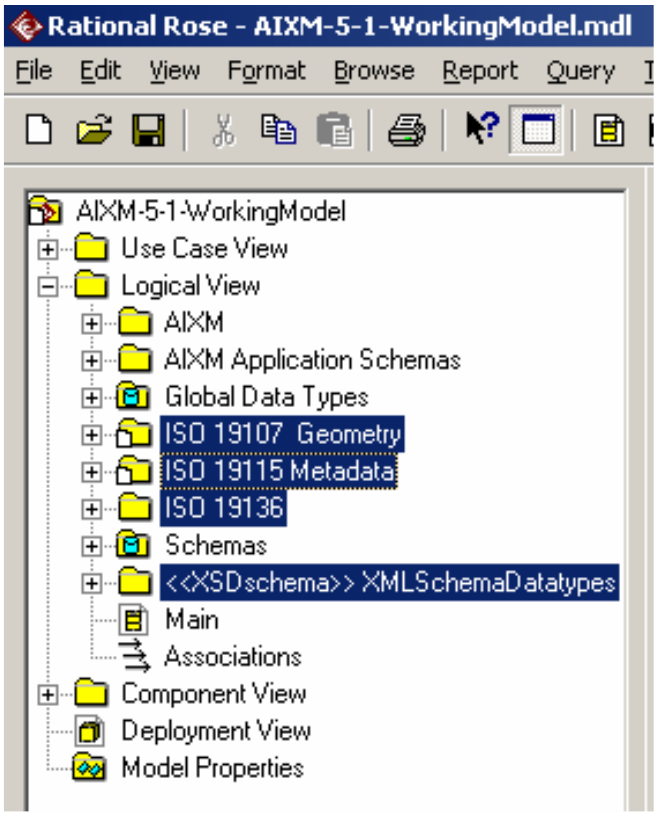
3.1.2 Metadata

The diagram also shows that each AIXM Feature and each TimeSlice is described by metadata. The AIXM XML schema incorporates the ISO 19139 metadata elements - see 3.2.2.

3.1.3 Extension

Finally, each TimeSlice may contain an Extension. The Extension mechanism allows each user of AIXM5 to define and use his own specific attributes and classes.

3.2 External packages



3.2.1 <<XSDschema>> XMLSchemaDatatypes

The XSD Schema Datatypes package declares XSD specific data types that are referenced by AIXM data types, when generating the AIXM XML (XSD) Schema. However, these XSD bindings do not mean that AIXM is "dependent" on the XML Schema specification. The pre-defined XSD simple types (such as string, decimal, unsignedInt, etc.) referenced by AIXM are sufficiently generic and mappable to the simple data types of many other data encoding standards.

3.2.2 ISO 19115 Metadata

This package contains some basic connections from the AIXM model to the ISO 19115 Metadata elements (MD_Metadata, MD_Constraints …).

3.2.3 ISO 19107 Geometry

This package contains some basic connections from the AIXM model to the ISO 19107 geometry elements (GM_Point, GM_Surface …).

3.2.4 ISO 19136

This package contains some basic connections from the AIXM model to GML specific elements, which are not part of the ISO 19107. Practically, the package contains only the data type NilReasonEnumeration, used to indicate the reason for a null value.

4 Mapping to the AIXM XML Schema

4.1 AIXM - core XSD files

The core AIXM exchange format is composed of three main files:

- AIXM_AbstractGML_ObjectTypes.xsd: the file references the ISO19139 Metadata Schema and defines the base AIXM Feature/Object constructs
  - AbstractAIXMFeatureType / AbstractAIXMFeature
  - AbstractAIXMTimesliceType / AbstractAIXMTimeslice
  - AbstractAIXMObjectType
  - AbstractAIXMPropertyType, which defines the nilReason for all the AIXM complex properties
- AIXM_Datatypes.xsd: this file contains the XML representation of all the data types defined in the AIXM UML model.
- AIXM_Features.xsd: this file contains the XML representation of all the AIXM features with all their properties (simple and complex).

The chapters here after specify the rules that govern the mapping between the AIXM UML model and the AIXM XML Schema.

4.2 AIXM is GML

The AIXM exchange model is an XML exchange standard based on a subset of the Geography Markup Language (GML). Essentially:

- AIXM Features are GML features;
- AIXM Objects are GML objects;
- AIXM follows the GML object-property concept.

4.3 The GML Object-Property Model

The GML object-property model explains some of the complexity of the AIXM UML to XSD mapping. It means that no GML object may appear as the immediate child of a GML object. Consequently, no element may be both a GML object and a GML property.

The object-property model prohibits the encoding of an object directly inside a feature, e.g.

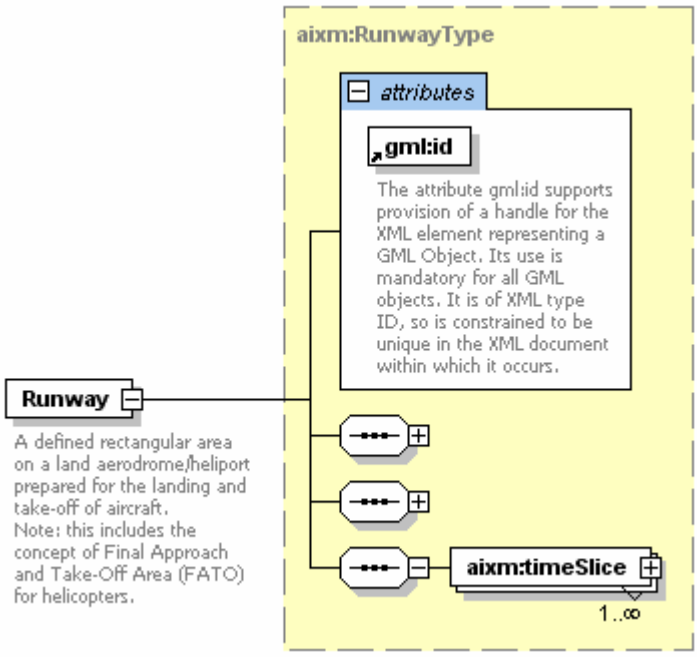
```
<AirportHeliport> <!-- feature -->
 <ElevatedPoint> <!-- object -->
```

Instead, in a compliant GML application schema, an association between two features (or a feature and an object) is implemented over a property of the feature, e.g.

```
<AirportHeliport> <!-- feature -->
 <hasReferencePoint> <!-- property -->
  <ElevatedPoint> <!-- object -->
```

The direction of the association arrow from the UML diagrams (the navigability) dictates which of the two association partners has the property that associates the other.

In the AIXM XML Schema, the object-property model is encoded by declaring a type and then assigning properties (attributes and relationships) to that type. The type defines the object.



4.4 Mapping Inheritance

Within the AIXM XML Schema, inheritance 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.

4.5 Mapping Name of Classes

The UML class name is used for the element names in the XML Schema.

4.6 Mapping Features

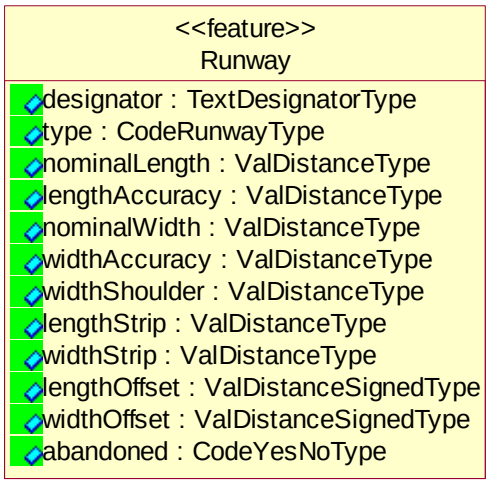
For each AIXM Feature in the UML, the following XML schema entities are created:

- *Feature*PropertyType
- *Feature*
- *Feature*Type
- *Feature*TimeSlicePropertyType
- *Feature*TimeSlice.
- *Feature*TimeSliceType
- *Feature*PropertyGroup
- Abstract*Feature*Extension

The direction in which the different types and elements are used in the schema definition (e.g. Feature uses FeatureType)

4.6.1 An Example Mapping

The Runway feature (shown below) will be used to illustrate the mapping. The example will concentrate on the properties (shown as attributes).



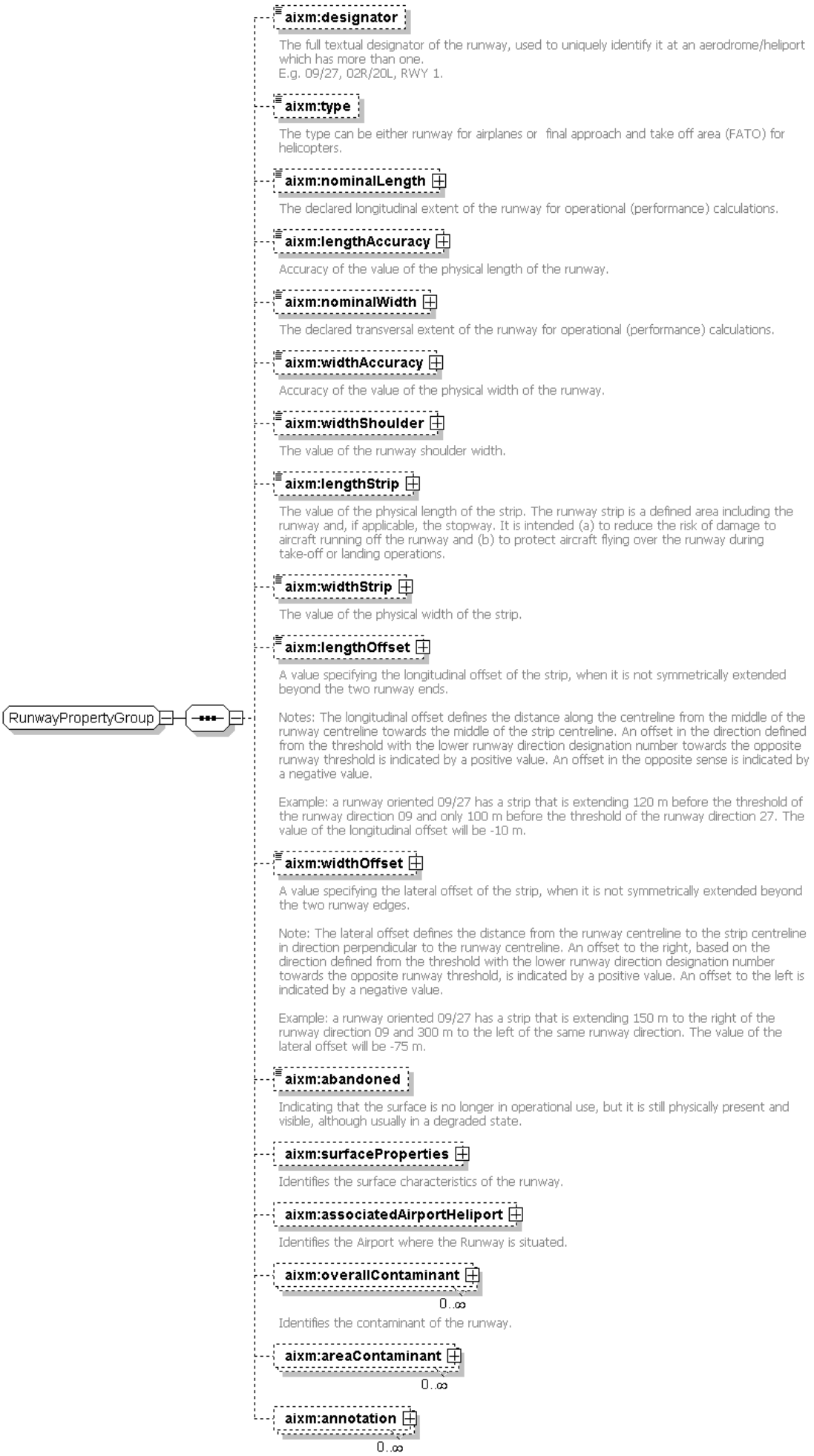
4.6.1.1 RunwayPropertyGroup

An XML Schema (XSD) group is generated for each feature containing all of the properties (attributes and relationships) of the feature.

The order in which the chilled elements of the group are declared is the following:

1. (if applicable) in the case of derived classes only, the property group of the super class is inserted first;
2. then, all the elements that correspond to class attributes, in the order that they appear in the UML class diagram
3. then, all the elements that correspond to association role names, in random order;
4. last the “annotation” property – note that for derived classes this property is only defined in the super class, therefore it will appear in the property group of the super class.

Below is an example of the RunwayPropertyGroup in graphic form and as an extract from the XSD. It shows clearly how the attributes are mapped from the UML to the XSD and how the relationship ‘associatedAirportHeliport’ is created.

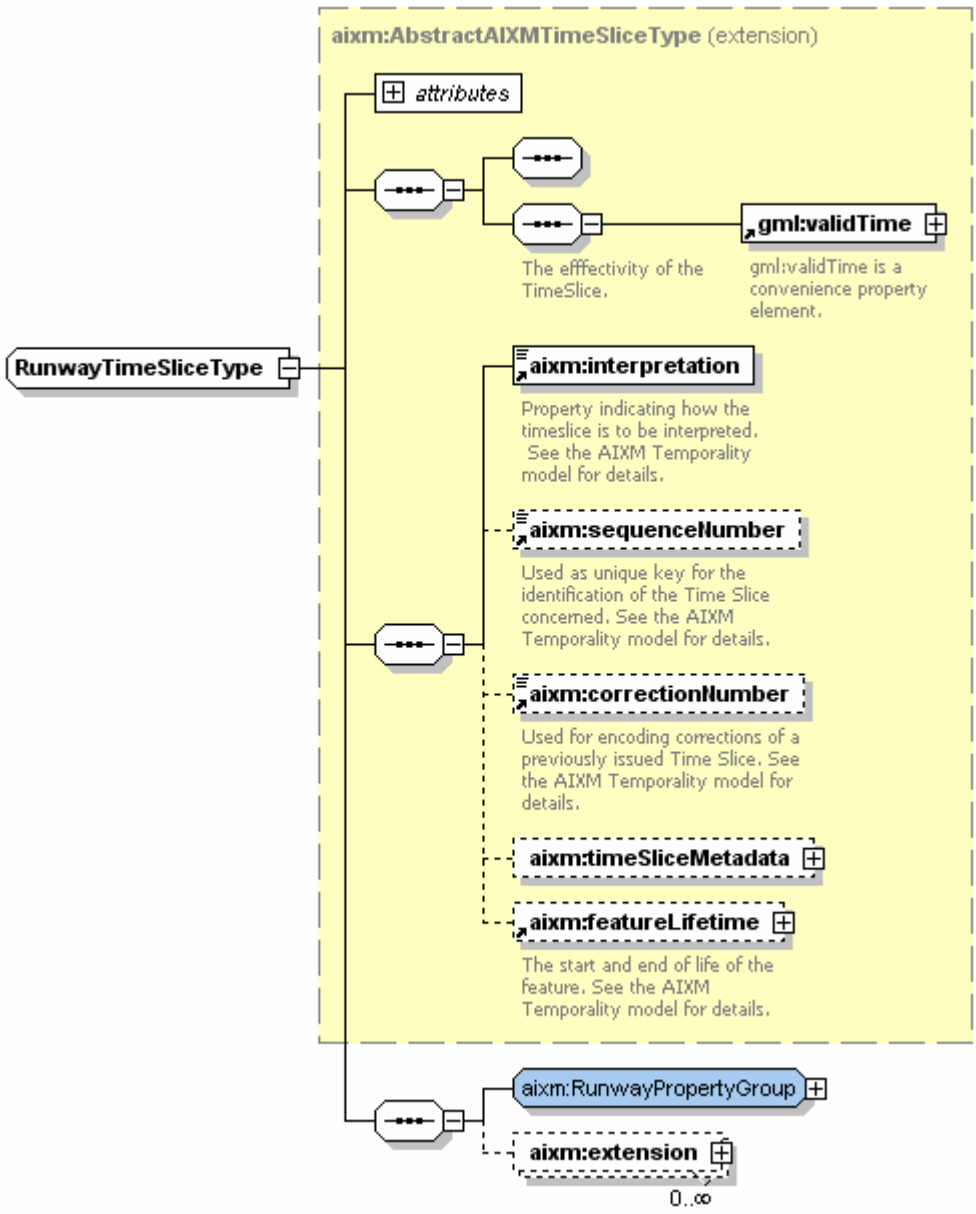

RunwayPropertyGroup
aixm:designator
The full textual designator of the runway, used to uniquely identify it at an aerodrome/heliport which has more than one.
E.g. 09/27, 02R/20L, RWY 1.
aixm:type
The type can be either runway for airplanes or final approach and take off area (FATO) for helicopters.
aixm:nominalLength
The declared longitudinal extent of the runway for operational (performance) calculations.
aixm:lengthAccuracy
Accuracy of the value of the physical length of the runway.
aixm:nominalWidth
The declared transversal extent of the runway for operational (performance) calculations.
aixm:widthAccuracy
Accuracy of the value of the physical width of the runway.
aixm:widthShoulder
The value of the runway shoulder width.
aixm:lengthStrip
The value of the physical length of the strip. The runway strip is a defined area including the runway and, if applicable, the stopway. It is intended (a) to reduce the risk of damage to aircraft running off the runway and (b) to protect aircraft flying over the runway during take-off or landing operations.
aixm:widthStrip
The value of the physical width of the strip.
aixm:lengthOffset
A value specifying the longitudinal offset of the strip, when it is not symmetrically extended beyond the two runway ends.
Notes: The longitudinal offset defines the distance along the centreline from the middle of the runway centreline towards the middle of the strip centreline. An offset in the direction defined from the threshold with the lower runway direction designation number towards the opposite runway threshold is indicated by a positive value. An offset in the opposite sense is indicated by a negative value.
Example: a runway oriented 09/27 has a strip that is extending 120 m before the threshold of the runway direction 09 and only 100 m before the threshold of the runway direction 27. The value of the longitudinal offset will be -10 m.
aixm:widthOffset
A value specifying the lateral offset of the strip, when it is not symmetrically extended beyond the two runway edges.
Note: The lateral offset defines the distance from the runway centreline to the strip centreline in direction perpendicular to the runway centreline. An offset to the right, based on the direction defined from the threshold with the lower runway direction designation number towards the opposite runway threshold, is indicated by a positive value. An offset to the left is indicated by a negative value.
Example: a runway oriented 09/27 has a strip that is extending 150 m to the right of the runway direction 09 and 300 m to the left of the same runway direction. The value of the lateral offset will be -75 m.
aixm:abandoned
Indicating that the surface is no longer in operational use, but it is still physically present and visible, although usually in a degraded state.
aixm:surfaceProperties
Identifies the surface characteristics of the runway.
aixm:associatedAirportHeliport
Identifies the Airport where the Runway is situated.
aixm:overallContaminant
0..∞
Identifies the contaminant of the runway.
aixm:areaContaminant
0..∞
aixm:annotation
0..∞
Generated by XmlSpy
www.altova.com

```
<group name="RunwayPropertyGroup">
   <sequence>
     <element name="designator" type="aixm:TextDesignatorType"
nillable="true" minOccurs="0"/>
     <element name="type" type="aixm:CodeRunwayType" nillable="true"
minOccurs="0"/>
     <element name="nominalLength" type="aixm:ValDistanceType"
nillable="true" minOccurs="0"/>
     <element name="lengthAccuracy" type="aixm:ValDistanceType"
nillable="true" minOccurs="0"/>
     ...
     <element name="associatedAirportHeliport"
type="aixm:AirportHeliportPropertyType" nillable="true" minOccurs="0"/>
     ...
     <element name="areaContaminant" type="aixm:
RunwayContaminationPropertyType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
     <element name="annotation" type="aixm:NotePropertyType" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
</group>
```

4.6.1.2 RunwayTimeSliceType

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.

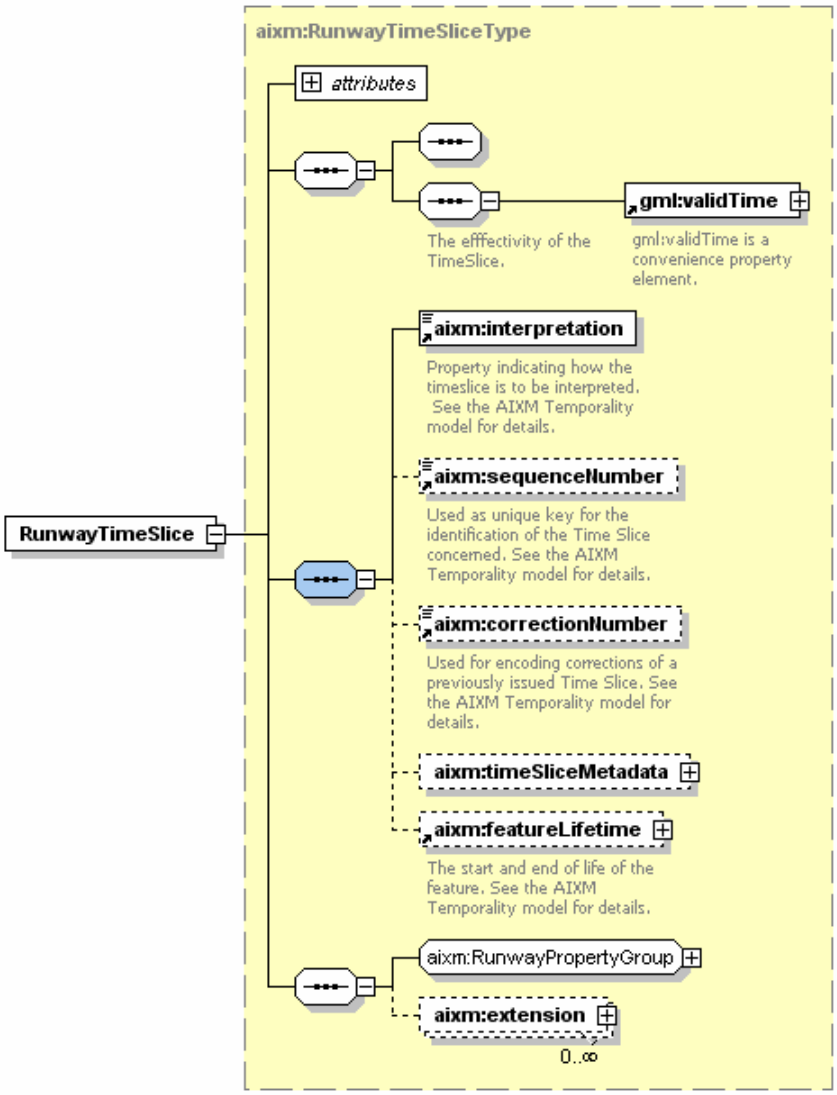
For each feature a TimeSlice property is created that contains an array of feature TimeSlice objects. This example shows the RunwayTimeSliceType encapsulating all of the Runway properties (RunwayPropertyGroup created above) that change over time.




```
<complexType name="RunwayTimeSliceType">
 <complexContent>
  <extension base="aixm:AbstractAIXMTimeSliceType">
   <sequence>
    <group ref="aixm:RunwayPropertyGroup"/>
    <element name="extension" minOccurs="0" maxOccurs="unbounded">
     <complexType>
      <sequence>
       <element ref="aixm:AbstractRunwayExtension"/>
      </sequence>
      <attributeGroup ref="gml:OwnershipAttributeGroup"/>
     </complexType>
    </element>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

4.6.1.3 RunwayTimeSlice

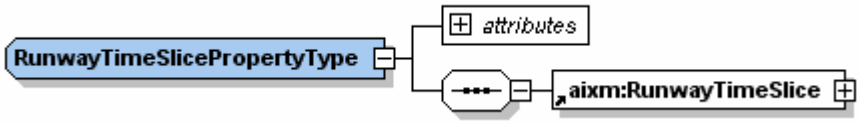
The *Feature*TimeSlice object is of type *Feature*TimeSliceType. Continuing the example, the RunwayTimeSlice element is of type RunwayTimeSliceType.




```
<element name="RunwayTimeSlice" type="aixm:RunwayTimeSliceType"
substitutionGroup="gml:AbstractTimeSlice"/>
```

4.6.1.4 RunwayTimeSlicePropertyType

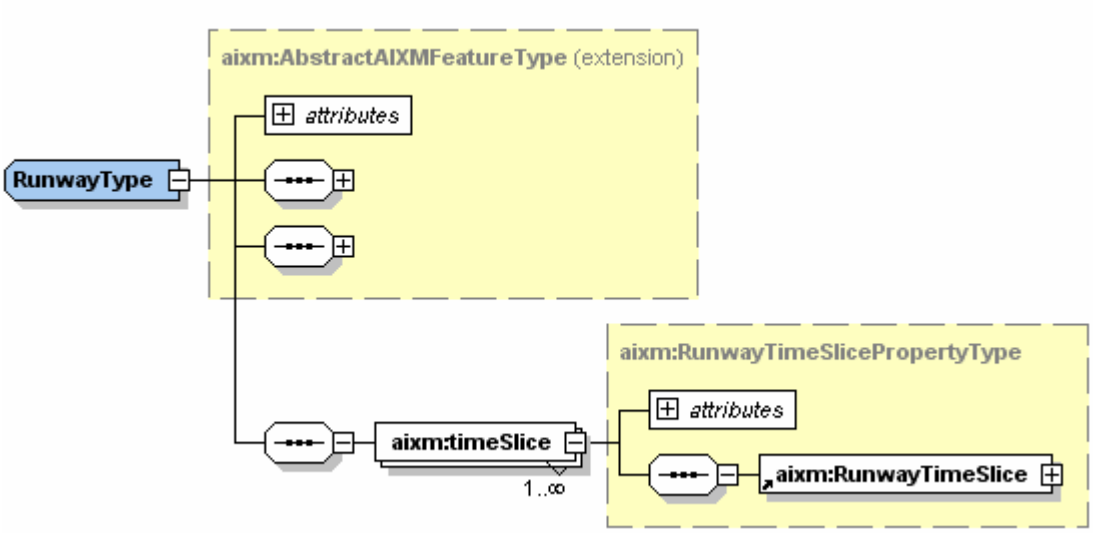
A GML property type containing a *Feature*TimeSlice objects is created.




```
<complexType name="RunwayTimeSlicePropertyType">
 <sequence>
  <element ref="aixm:RunwayTimeSlice"/>
 </sequence>
 <attributeGroup ref="gml:OwnershipAttributeGroup"/>
</complexType>
```

4.6.1.5 RunwayType

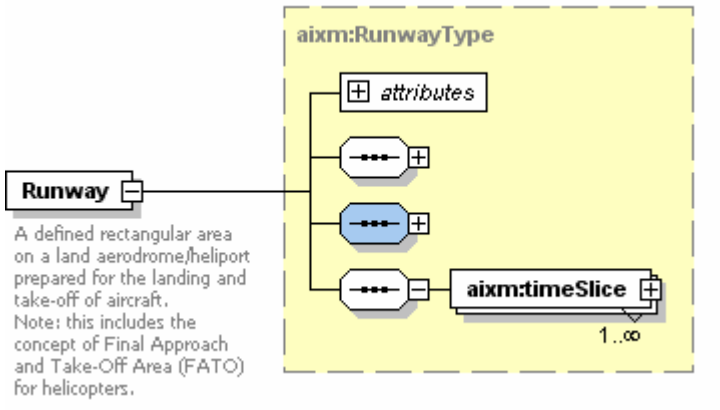
Continuing with the object-property model, the Runway feature type is created extending the AbstractAIXMFeatureType with the RunwayTimeSlice object created above.




```
<complexType name="RunwayType">
 <complexContent>
  <extension base="aixm:AbstractAIXMFeatureType">
   <sequence>
    <element name="timeSlice" type="aixm:RunwayTimeSlicePropertyType"
maxOccurs="unbounded"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

4.6.1.6 Runway

The Runway feature is then defined by the RunwayType.




```
<element name="Runway" type="aixm:RunwayType"
substitutionGroup="aixm:AbstractAIXMFeature">
 <annotation>
  <appinfo>RWY</appinfo>
  <appinfo><gml:description>A defined rectangular area on a land
aerodrome/heliport prepared for the landing and take-off of aircraft. Note:
this includes the concept of Final Approach and Take-Off Area (FATO) for
helicopters.</gml:description></appinfo>
 </annotation>
</element>
```

4.6.1.6.1 RunwayPropertyType

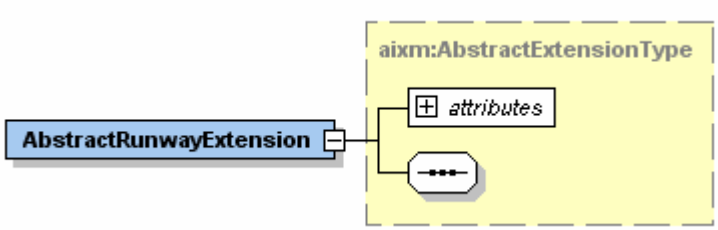
When a property of a feature is a relationship, the relationship must be associated to another feature or object. This is done through the creation of the *Feature*PropertyType, in this case, the RunwayPropertyType.

In AIXM, when the relationship or association points to another feature, the feature is referenced using the `xlink:href` attribute (it's always a "remote" encoding). When the relationship points to an object, the object is included inside the parent. (Objects cannot exist without the parent.) Since a Runway is a feature the RunwayPropertyType is created with the attribute xlink:href .


```
<complexType name="RunwayPropertyType">
 <attributeGroup ref="gml:OwnershipAttributeGroup"/>
 <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>
```

4.6.1.7 RunwayExtension

All Features and Objects can be extended. A relationship is created with an abstract XML element that acts as the root for all extensions. Below is an example of the extension for the Runway feature. The AbstractRunwayExtension element uses the AbstractExtensionType as shown below.




```
<element name="AbstractRunwayExtension" type="aixm:AbstractExtensionType"
abstract="true" substitutionGroup="aixm:AbstractExtension"/>
```

4.7 Mapping Objects

AIXM objects are encoded as GML objects. For the most part, the XML schema entities are created in the same way as for Features, following the object-property model. However it is important to remember that AIXM objects do not exist outside of a feature and are therefore part of the feature timeslice. TimeSlice types and elements are not created for objects.

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

- *Object*PropertyGroup
- *Object*
- *Object*Type
- *Object*PropertyType
- Abstract*Object*Extension

*Object*Type is complex type which extends AbstracAIXMObjectType.

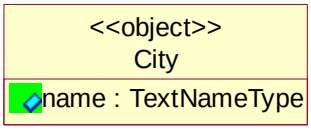
```
<complexType name="NavaidEquipmentDistanceType">
 <complexContent>
  <extension base="aixm:AbstractAIXMObjectType">
   <sequence>
    <group ref="aixm:NavaidEquipmentDistancePropertyGroup"/>
    <element name="extension" minOccurs="0" maxOccurs="unbounded">
     <complexType>
      <sequence>
       <element ref="aixm:AbstractNavaidEquipmentDistanceExtension"/>
      </sequence>
      <attributeGroup ref="gml:OwnershipAttributeGroup"/>
     </complexType>
    </element>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

*Object*PropertyType type is a complex type which extends aixm:AbstractAIXMPropertyType.

```
<complexType name="NavaidEquipmentDistancePropertyType">
 <complexContent>
  <extension base="aixm:AbstractAIXMPropertyType">
   <sequence>
    <element ref="aixm:NavaidEquipmentDistance"/>
   </sequence>
   <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </extension>
 </complexContent>
</complexType>
```

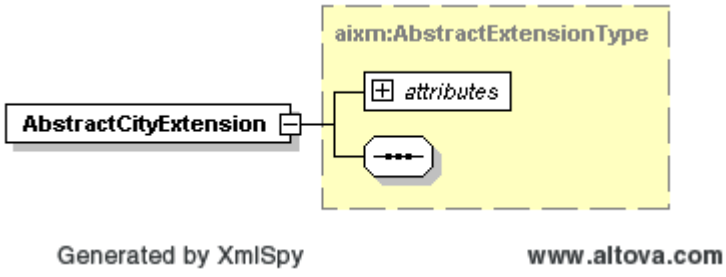
4.7.1 An Example Mapping

The example will use the City object illustrated below. The object represents the town that is served by an AirportHeliport.



4.7.1.1 AbstractCityExtension

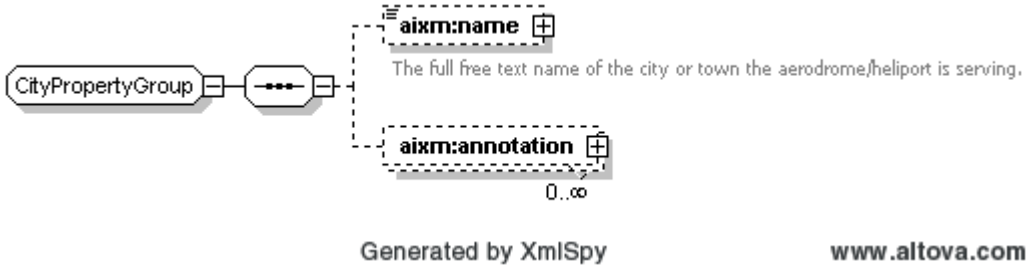
An abstract XML element acts as the root for all extension to the City object. Object extensions are defined in the same way as Feature extensions.




```
<element name="AbstractCityExtension" type="aixm:AbstractExtensionType"
abstract="true" substitutionGroup="aixm:AbstractExtension"/>
```

4.7.1.2 CityPropertyGroup

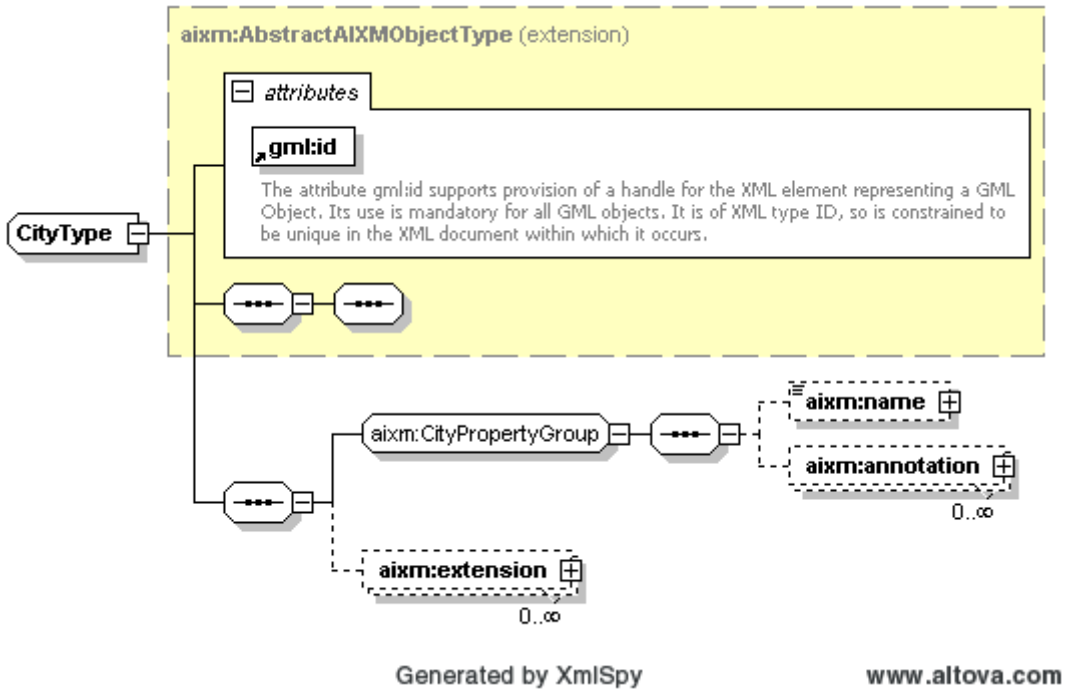
An XSD group containing the properties of the City object is created, again similar to Features.




```
<group name="CityPropertyGroup">
 <sequence>
  <element name="name" type="aixm:TextNameType" nillable="true"
minOccurs="0">
   <annotation>
    <appinfo>AIXM 4.5</appinfo>
    <appinfo><gml:description>The full free text name of the city or town
the aerodrome/heliport is serving.
    </gml:description></appinfo>
   </annotation>
  </element>
  <element name="annotation" type="aixm:NotePropertyType" nillable="true"
minOccurs="0" maxOccurs="unbounded">
  </element>
 </sequence>
</group>
```

4.7.1.3 CityType

The CityType definition uses the CityPropertyGroup and the extension. It extends AbstractAIXMObjectType

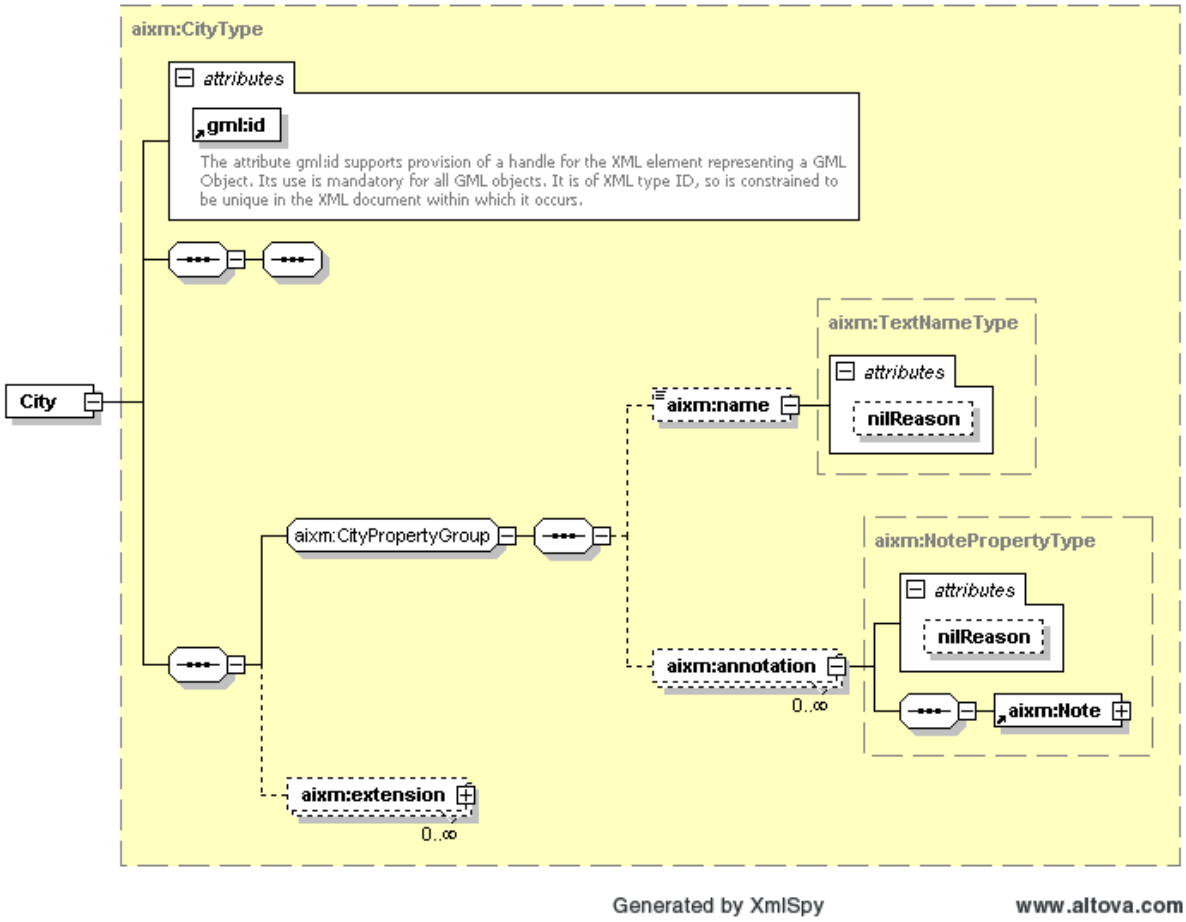



```
 <complexType name="CityType">
  <complexContent>
   <extension base="aixm:AbstractAIXMObjectType">
    <sequence>
```

```
      <group ref="aixm:CityPropertyGroup"/>
      <element name="extension" minOccurs="0" maxOccurs="unbounded">
       <complexType>
        <sequence>
         <element ref="aixm:AbstractCityExtension"/>
        </sequence>
        <attributeGroup ref="gml:OwnershipAttributeGroup"/>
       </complexType>
      </element>
     </sequence>
    </extension>
   </complexContent>
  </complexType>
```

4.7.1.4 City

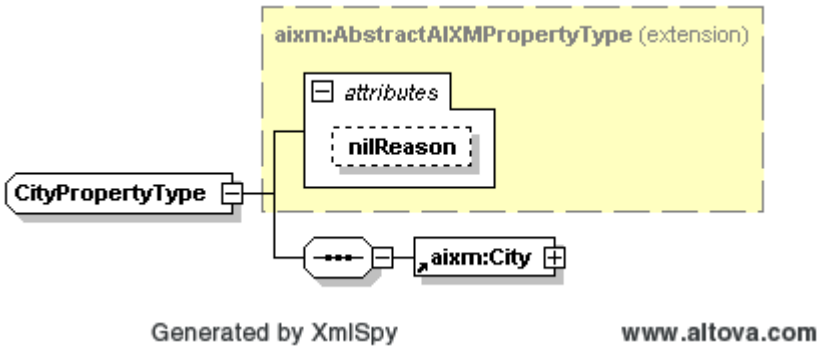
The City object is then defined as an XSD element of type CityType.




```
<element name="City" type="aixm:CityType">
 <annotation>
  <appinfo><gml:description>A city or location that may be served by an
airport/heliport.</gml:description></appinfo>
 </annotation>
</element>
```

4.7.1.5 CityPropertyType

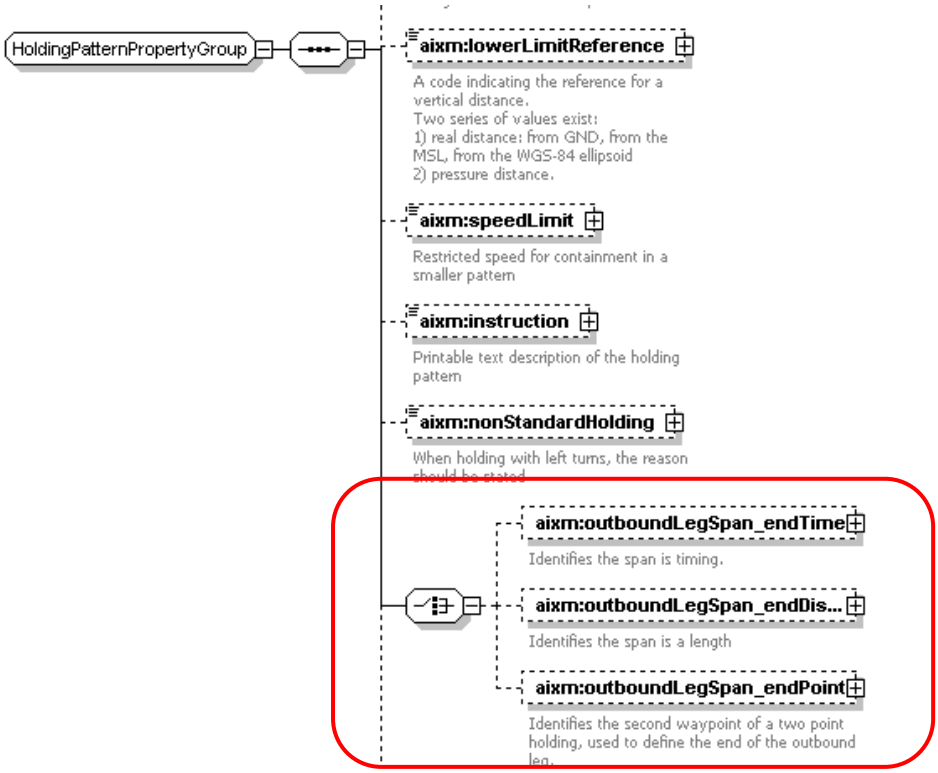
An XSD complex type representing a GML property type is created. A Feature uses this element to include the City object rather than reference it (using `xlink:href`) because object does not exist without the parent.




```
<complexType name="CityPropertyType">
  <complexContent>
    <extension base="aixm:AbstractAIXMObjectType">
      <sequence>
        <element ref="aixm:City"/>
      </sequence>
      <attributeGroup ref="gml:OwnershipAttributeGroup"/>
    </extension>
  </complexContent>
</complexType>
```

4.8 Mapping Choices

Classes marked with the stereotype <<choice>> do not appear in the XML Schema. Instead, the choice of elements is created.



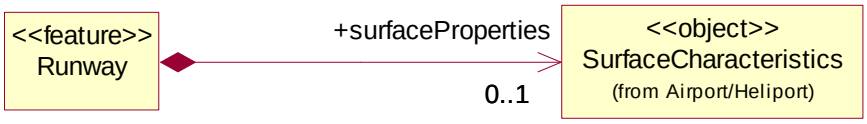
The name of the element is the concatenation of the role of the <<choice>> class with the role of the target class of each choice branch, separated by "_".

```
<group name="HoldingPatternPropertyGroup">
  <sequence>
    ............
     <element name="nonStandardHolding" type="aixm:CodeYesNoType"
nillable="true" minOccurs="0">
```

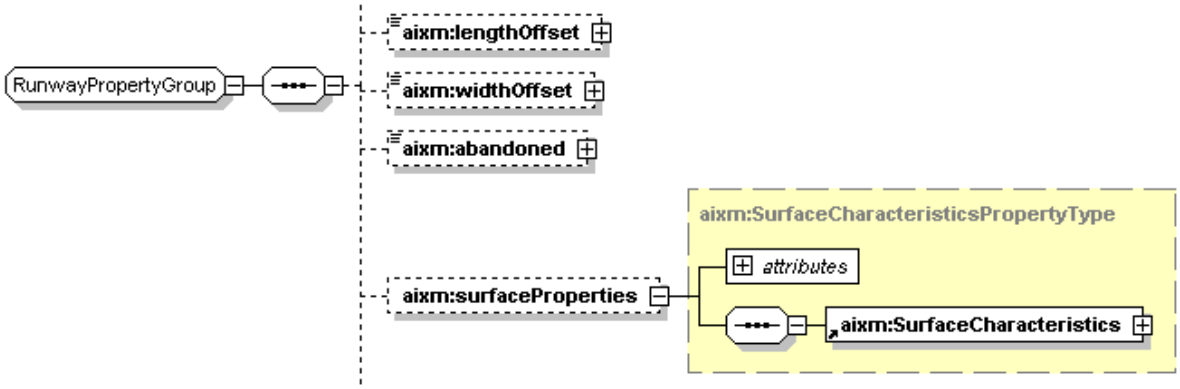
```
      <annotation>
        <appinfo>
          <gml:description>ndicates whether the HoldingPattern is non-
standard, for example because it uses left-hand turns.</gml:description>
        </appinfo>
      </annotation>
    </element>
    <choice>
      <element name="outboundLegSpan_endTime"
type="aixm:HoldingPatternDurationPropertyType" nillable="true"
minOccurs="0">
        <annotation>
          <appinfo>
            <gml:description>Span is timing</gml:description>
          </appinfo>
        </annotation>
      </element>
      <element name="outboundLegSpan_endDistance"
type="aixm:HoldingPatternDistancePropertyType" nillable="true"
minOccurs="0">
        <annotation>
          <appinfo>
            <gml:description>span is length</gml:description>
          </appinfo>
        </annotation>
      </element>
      <element name="outboundLegSpan_endPoint"
type="aixm:SegmentPointPropertyType" nillable="true" minOccurs="0">
        <annotation>
          <appinfo>
            <gml:description>The second waypoint of a two point holding, used
to define the end of the outbound leg.</gml:description>
          </appinfo>
        </annotation>
      </element>
    </choice>
   ............
  </sequence>
</group>
```

4.9 Mapping Relationships to Objects

Relationships are encoded by creating an XML element with the same name as the role name on the UML model. It is of type *Object*PropertyType.



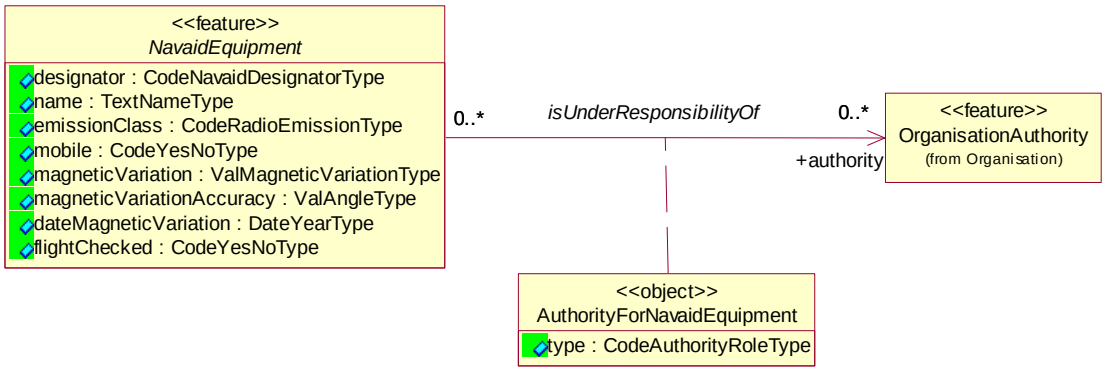
In this example, the SurfaceCharacterisitcs object is a property of the Runway. The “surfaceProperties” property of the Runway is defined as being of type SurfaceCharacteristicsPropertyType.



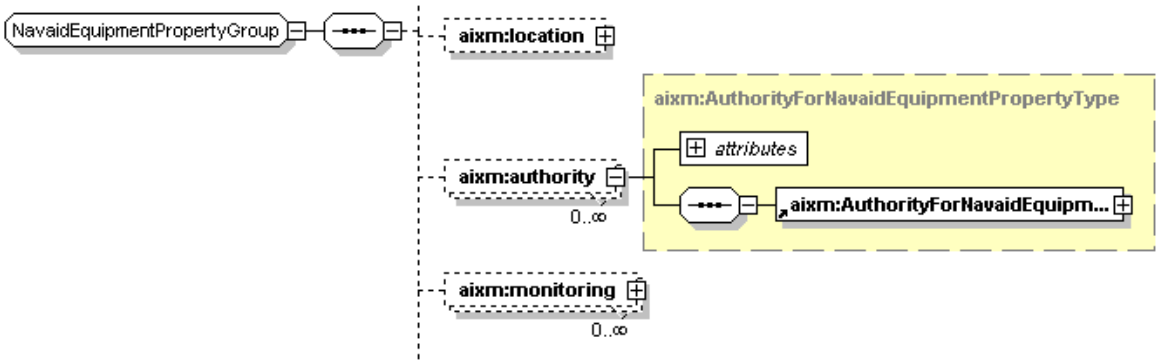

```
<element name="surfaceProperties"
type="aixm:SurfaceCharacteristicsPropertyType" minOccurs="0"/>
```

4.9.1 Mapping Associations with Association Classes

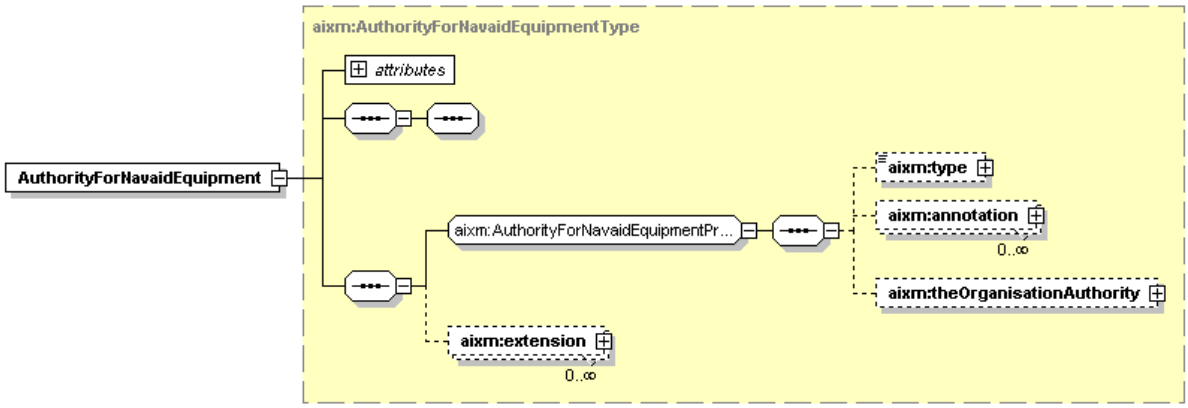
In the UML below, the NavaidEquipment feature has a relationship to the OrganisationAuthority feature. This relationship contains properties defined in the AuthorityForNavaidEquipment class.



When mapping this in XSD, an 'authorityForNavaidEquipment' property is created in the NavaidEquipment feature as shown below. The name of this property is automatically derived from the name of the association class, by conversion to lowerCamelCase style. The direction of the arrow is important. If the direction would have been to the NavaidEquipment, the property would have been created in the OrganisationAuthority feature.

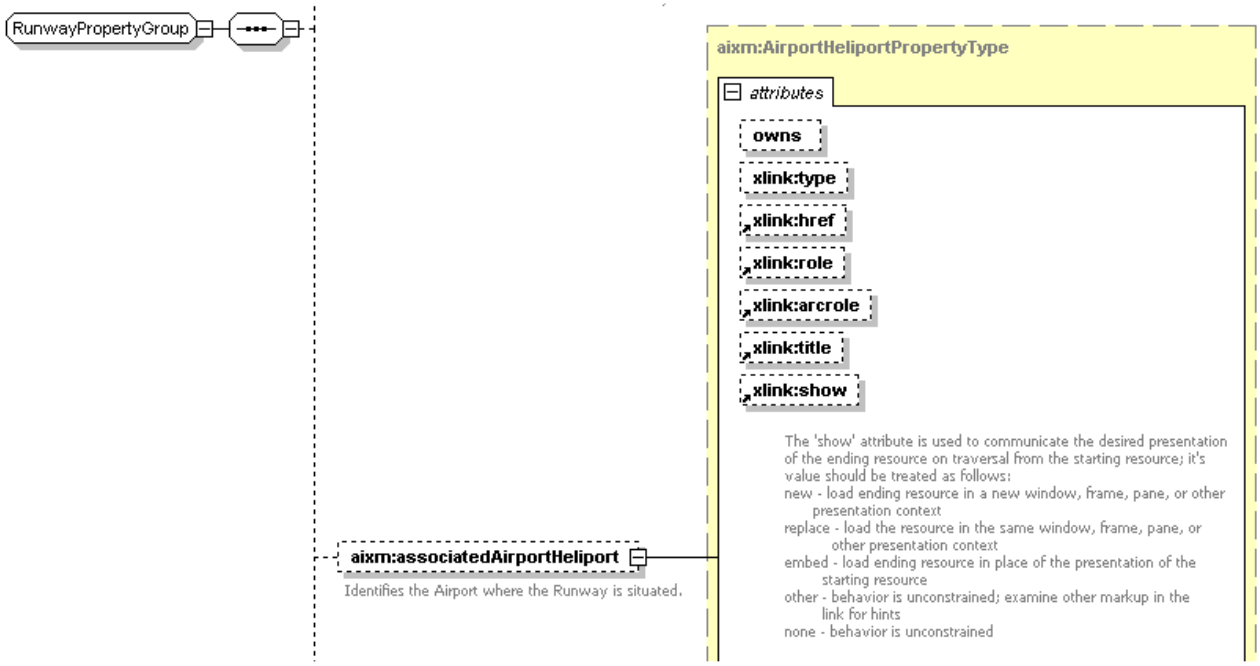


A second step is then required to complete the XSD. In this case an element named 'theOrganisationAuthority' is added in the definition of the AuthorityForNavaidEquipmentPropertyGroup, based on the role of the OrganisationAuthority class in this association.



4.10 Mapping Relationships to Features

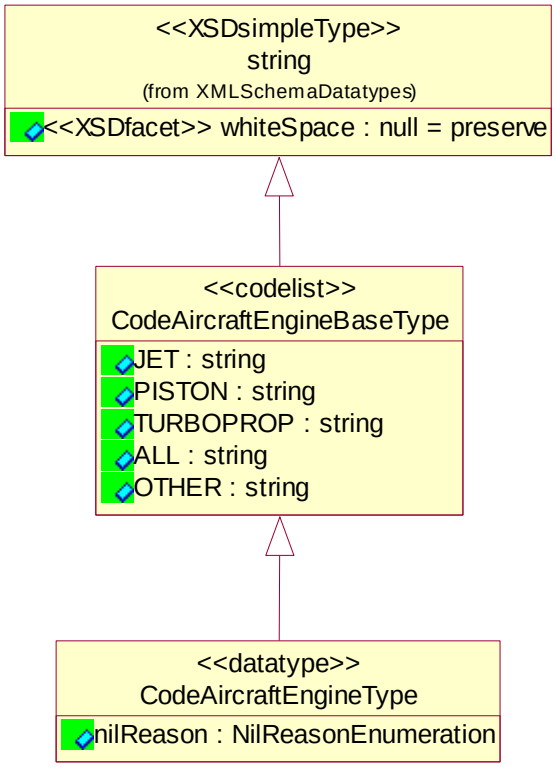
In AIXM, Relationships to features are described by reference using `xlink:href`. The UML role name is used for the XML element name and the XML element is of type *Feature*PropertyType.



4.11 Mapping Data Types

4.11.1 <<codelist>>

Codelists are given by the stereotype <<codelist>>. As it can be seen from the diagram elow, for each <<codelist>> type, there also is a <<datatype>> class, which defines the nilReason attribute.



First, the <<codelist>> class is converted into a simpleType in the XSD:

```
<simpleType name="CodeAircraftEngineBaseType">
   <annotation>
    <appinfo><gml:description>A code indicating the type of aircraft engine
(for example, jet, piston, turbo).</gml:description></appinfo>
   </annotation>
   <union>
    <simpleType>
     <restriction base="xsd:string">
      <enumeration value="JET">
       <annotation>
        <appinfo><gml:description>Jet Engine</gml:description></appinfo>
       </annotation>
      </enumeration>
      <enumeration value="PISTON">
       <annotation>
        <appinfo><gml:description>Piston
Engine</gml:description></appinfo>
       </annotation>
      </enumeration>
      <enumeration value="TURBOPROP">
       <annotation>
        <appinfo><gml:description>Turbo Propeller
Engine</gml:description></appinfo>
       </annotation>
      </enumeration>
```

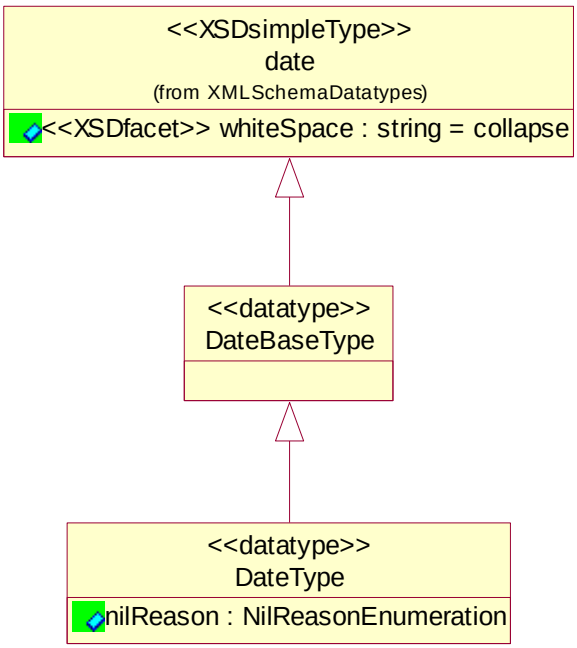
```
       <enumeration value="ALL">
        <annotation>
         <appinfo><gml:description>All aircraft engine
types.</gml:description></appinfo>
        </annotation>
       </enumeration>
     </restriction>
    </simpleType>
    <simpleType>
     <restriction base="string">
      <pattern value="OTHER(:(\w|_){1,58})?"/>
     </restriction>
    </simpleType>
   </union>
  </simpleType>
```

Note that the simple data types is declared as a union between the enumerated values declared in the UML model (with the exception of the value “OTHER”) and a string with the pattern “OTHER(:(\w|_){1,58})?”. This enables <<codelist>> data types to include values that are not supported by the enumeration list. For example, an electric engine type could be encoded as “OTHER:ELECTRIC”.

In addition, a complex type is defined, including the declaration of the nilReason attribute:

```
  <complexType name="CodeAircraftEngineType">
   <simpleContent>
    <extension base="aixm:CodeAircraftEngineBaseType">
     <attribute name="nilReason" type="gml:NilReasonEnumeration"/>
    </extension>
   </simpleContent>
  </complexType>
```

4.11.2 <<datatype>> - default case



As for <<codeList>>, the mapping of <<datatype>> used to type simple properties (see 2.7.1.1) consists of two steps.

The first step is the creation of the simpleType corresponding to the BaseType.

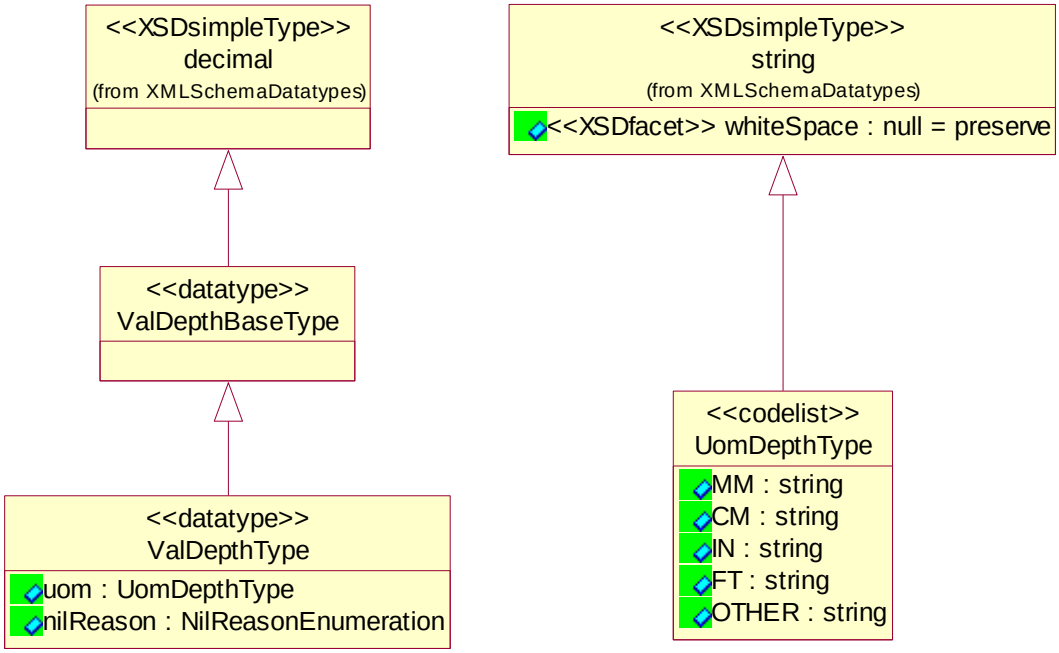
```
<simpleType name="DateBaseType">
 <restriction base="xsd:date">
 </restriction>
</simpleType>
```

The second step is the creation of the complexType which defines the attribute nilReason.

```
 <complexType name="DateType">
  <simpleContent>
   <extension base="aixm:DateBaseType">
    <attribute name="nilReason" type="gml:NilReasonEnumeration "/>
   </extension>
  </simpleContent>
 </complexType>
```

4.11.3 <<datatype>> with Unit of Measurement

A Unit of measurement (UOM) exists for many data types that take numerical values. This has been modelled as a uom attribute in the <<datatype>> class.



The XSD mapping of uom types follows the same rules as for any other <<codelist>>, except that no complex type is required with the nilReason.

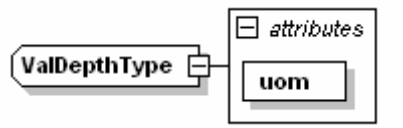
```
 <simpleType name="UomDepthType">
  <union>
   <simpleType>
    <restriction base="xsd:string">
     <enumeration value="MM">
     </enumeration>
     <enumeration value="CM">
     </enumeration>
     <enumeration value="IN">
     </enumeration>
     <enumeration value="FT">
     </enumeration>
    </restriction>
   </simpleType>
   <simpleType>
```

```
      <restriction base="string">
        <pattern value="OTHER:\w{2,58}"/>
      </restriction>
    </simpleType>
  </union>
 </simpleType>
```

In a second step, the class ValDepthBaseType is generated as a simple type, as described in 4.11.2.

```
<simpleType name="ValDepthBaseType">
 <restriction base="xsd:decimal"/>
</simpleType>
```

Then, the uom attribute is added to the complexType ValDepthType, after the definition of nilReason attribute.



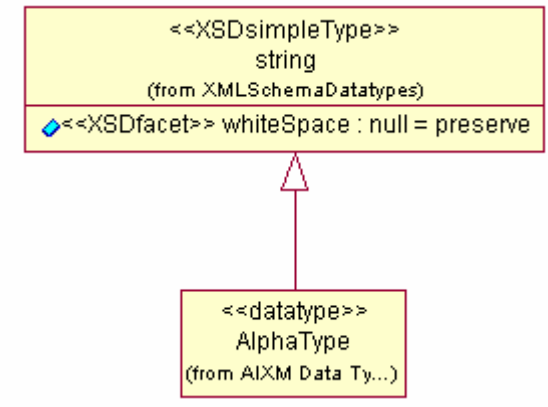

```
<complexType name="ValDepthType">
 <simpleContent>
  <extension base="aixm:ValDepthBaseType">
   <attribute name="nilReason" type="gml:NilReasonEnumeration"/>
   <attribute name="uom" type="aixm:UomDepthType" use="required"/>
  </extension>
 </simpleContent>
</complexType>
```

4.11.4 Particular cases

4.11.4.1 <<datatype>> with no BaseType

The 5 data types listed in 2.7.1.1 map directly to the built-in datatypes defined by the XML schema specification. The default datatypes are `string`, `float`, `double`, etc, which are considered `simpleTypes`.

The AlphaType acts as a convenient example.




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

4.11.4.2 <<datatype>> XHTMLBaseType

<<datatype>> XHTMLBaseType represents a structured XHTML document compliant with http://www.w3.org/1999/xhtml. It should be mapped as follows in XML:

```
<complexType name="XHTMLBaseType">
  <sequence>
    <any namespace="http://www.w3.org/1999/xhtml" minOccurs="1"
maxOccurs="unbounded" processContents="skip"/>
  </sequence>
</complexType>
```