Document Title	Model Persistence Rules for XML
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	122
Document Classification	Standard

Document Version	2.2.0
Document Status	Final
Part of Release	3.1
Revision	5

Document Change History

Date	Version	Changed by	Change Description
30.09.2010	2.2.0	AUTOSAR Administration	• Updated default configuration of tagged values • Updated default configuration of multiplicities • Legal disclaimer revised
23.06.2008	2.1.3	AUTOSAR Administration	• Legal disclaimer revised
31.10.2007	2.1.2	AUTOSAR Administration	• Document meta information extended • Small layout adaptations made
24.01.2007	2.1.1	AUTOSAR Administration	• “Advice for users” revised • “Revision Information” added
05.12.2006	2.1.0	AUTOSAR Administration	• Updated instanceRef references • Only absolute paths allowed • Naming of instanceRef • Destination type of references • Version info in namespace • Legal disclaimer revised
17.05.2006	2.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction

The AUTOSAR metamodel describes all information entities which can be used to describe an AUTOSAR system. XML is chosen as a basis for the exchange of formal descriptions of AUTOSAR systems. This document describes how a W3C XML schema specification[8] compliant XML schema can be compiled out of the AUTOSAR metamodel [3] or any other metamodel that is modeled according to the AUTOSAR modeling guidelines [4]. Using the persistence rules a new XML schema can be generated automatically whenever the metamodel is updated.
The model persistence rules defined in this document exceed the configuration possibilities of comparable approaches like XMI [5][6] and enables the generic reproduction of a wide range of existing XML schemas out of well-structured UML models. The numbers in brackets you can find in this specification identify spefication items.

Figure 1-1 describes the model persistence rules for XML in the overall context. The metalevels of the AUTOSAR modeling approach are described on the left side of the image:

- The syntax and semantics of the language **UML2.0** is described on the meta metalevel (M3). The **AUTOSAR template profile** [3][4] defines, which parts of UML2.0 are allowed to be used in the AUTOSAR metamodel.
- The **AUTOSAR metamodel**[3] is a UML2.0[14] model that defines the language for describing AUTOSAR systems. The AUTOSAR metamodel is a graphical representation of a template. UML2.0 class diagrams are used to describe the attributes and their interrelationships. Stereotypes and OCL[16] (object constraint language) are used for defining specific semantics and constraints.
- An **AUTOSAR model** is an instance of the AUTOSAR metamodel. The information contained in the AUTOSAR model can be anything that is representable according to the AUTOSAR metamodel.

The metalevels of the XML language are described on the right side of Figure 1-1:

- The **W3C XML schema specification** [8]defines how a W3C XML schema can be defined.
- The **AUTOSAR XML schema** is a W3C XML schema that defines the language for exchanging AUTOSAR models. This XML schema is derived from the AUTOSAR metamodel by means of the rules defined in this document. The AUTOSAR XML schema defines the AUTOSAR data exchange format.
- An **AUTOSAR XML description** describes the XML representation of an AUTOSAR model. The AUTOSAR XML description can consist of several fragments (e.g. files). Each individual fragment must validate successfully against the AUTOSAR XML schema.

This document describes how the AUTOSAR metamodel is mapped to the AUTOSAR XML schema by means of the **model persistence rules** for XML.

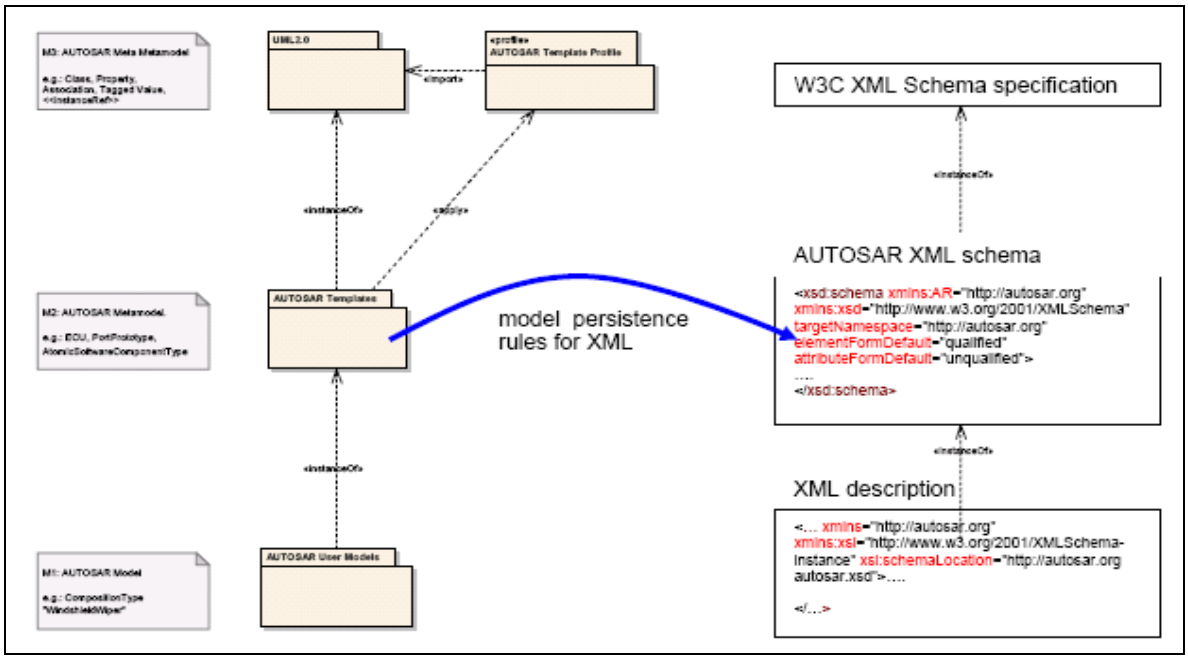


Figure 1-1: Context of Model Persistence Rules for XML

This document is structured as follows:

- Chapter 1 (this chapter) describes the model persistence rules for XML in the overall context of the AUTOSAR metamodel and the XML language.
- Chapter 2 traces the requirements on the model persistence rules for XML to specification items and chapter in this document.
- Chapter 3 describes the XML schema design principles. Some notes on the UML2.0 semantics of associations, attributes, references and properties are given first, followed by a discussion on the basic principles including aspects such as names of XML elements, transmitting incomplete data, linking, etc.
- Chapter 4 describes how the model persistence rules for XML can be parameterized by means of tagged values. Additionally a default configuration for mapping the AUTOSAR metamodel to the AUTOSAR XML schema is given.
- Chapter 5 describes the model persistence rules in more detail. The relationship between the rules is illustrated graphically.

Note: This document contains examples for illustration of the model persistence rules for XML. Some examples are taken out of the AUTOSAR metamodel and simplified for better readability. Therefore these examples might not be in sync with the latest version of the AUTOSAR metamodel.

2 Requirements tracing

The following table lists the requirements on the AUTOSAR XML schema and the AUTOSAR metamodel which are relevant for the AUTOSAR data exchange format as defined in [2]. The column “Satisfied by” depicts where a given requirement is covered in this document.

Requirement	***Satisfied by***
[ATI0019] AUTOSAR XML schema SHALL support for description of incomplete models and model fragments	3.9 and 4, 5
[ATI0020] AUTOSAR XML schema SHALL be based on proven XML design concepts	3, 4
[ATI0021] AUTOSAR SHOULD provide for upward compatibility detection	Not yet covered, planned for AUTOSAR phase 2
[ATI0023] AUTOSAR XML schema SHALL allow for flexible distribution of XML descriptions over several XML files and referencing between them	3.8, 5.5
[ATI0025] AUTOSAR XML schema SHALL be consistent with the AUTOSAR metamodel	3.2, 4.2
[ATI0027] AUTOSAR XML schema MAY use XML namespace	3, 4, 5
[ATI0028] AUTOSAR XML schema SHALL support for strict XML validation	4, 5
[ATI0029] AUTOSAR XML schema SHALL contain the version information of the metamodel it was generated from	3.10
[ATI0030] AUTOSAR XML schema SHALL ensure upwards compatibility of existing XML descriptions in case on minor changes in the metamodel	4.2
[ATI0031] AUTOSAR XML schema SHOULD provide extension mechanism	Not yet covered, planned for AUTOSAR phase 2
[ATI0032] AUTOSAR XML schema SHALL support for unambiguous mapping to metamodel instances	3.2, 4, 5

3 XML Schema design principles [Tc0003]

This chapter first describes some notes on XML schema and on UML2.0 semantics of the AUTOSAR metamodel and then gives a brief description on some basic principles, which include a short description of XML names, order of XML elements and linking.

3.1 Notes on UML2.0 semantics of the AUTOSAR metamodel

In UML2.0 [14] attributes and navigable association ends are represented as properties. Since the AUTOSAR Template Profile and Modeling Guide [4] only supports associations with two association ends, attributes and associations can be considered as equivalent for the persistence rules for XML. Therefore the persistence rules for XML can concentrate on classes and properties.

The following four sections give more information on the UML2.0 semantics of these concepts. Each chapter contains a diagram which shows the concept in the UML graphical notation (upper half of the diagram) and how it is represented as an instance of the UML2.0 metamodel (lower part of the diagram).

Please note that in UML2.0 compositions and references are both described by means of associations. The only difference is the value of the attribute “aggregation” of the association ends. For a more detailed description of the UML2.0 semantics please refer to the UML2.0 superstructure specification [14].
In the following sections the value of the attribute “aggregation” of the navigable association end is shown in brackets behind the association.

3.1.1 Representation of association (aggregation = composite) [Tc0003Sc1]

Figure 3-1 depicts how a composite association is represented by means of instances of the UML2.0 metamodel. The association end ‘theC’ is navigable from class A and is represented as a property that is owned by class ‘A’. The association end that is not navigable is owned by the association.
The information represented by the anonymous association and the not navigable property is not relevant for the model persistence rules: From the point of view of persistence rules there is not difference between composite association and an attribute (see also next section).

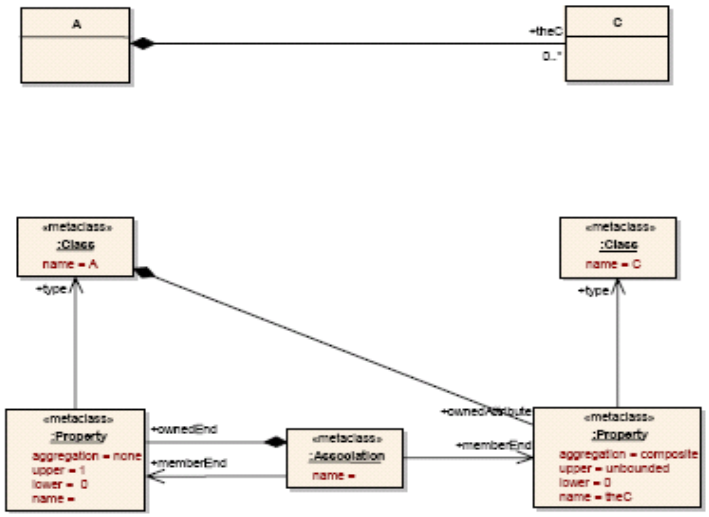


Figure 3-1: Representation of association (aggregation = composite)

3.1.2 Representation of attribute (aggregation = composite) [Tc0003Sc2]

Figure 3-2 depicts how an attribute is represented by means of instances of the UML2.0 metamodel. The attribute ‘theC’ is represented as a property that is owned by class ‘A’.

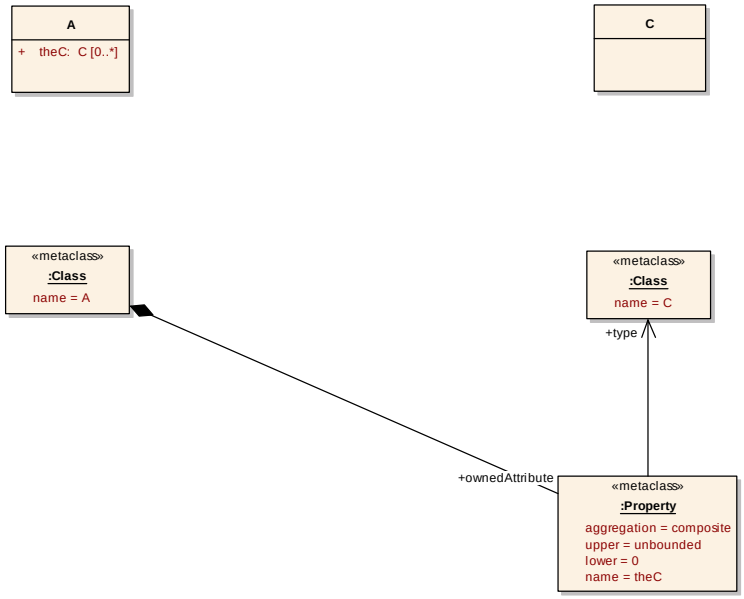


Figure 3-2: Representation of attribute (aggregation = composite)

3.1.3 Representation of association (aggregation = none) [Tc0003Sc3]

Figure 3-3 depicts how a reference (association with aggregation = none) is represented by means of instances of the UML2.0 metamodel. The association end ‘theB’ is navigable from class D and is represented as a property that is owned by class ‘D’.

The information represented by the anonymous association and the not navigable property is not relevant for the model persistence rules. From the point of view of the persistence rules there is no difference between a reference and an attribute with aggregation=none (see also next section). However, the AUTOSAR metamodel assigns the stereotypes <<instanceRef>> and <<isOfType>> to references. This special semantics need to be handled separately as described in section 4.2.3.

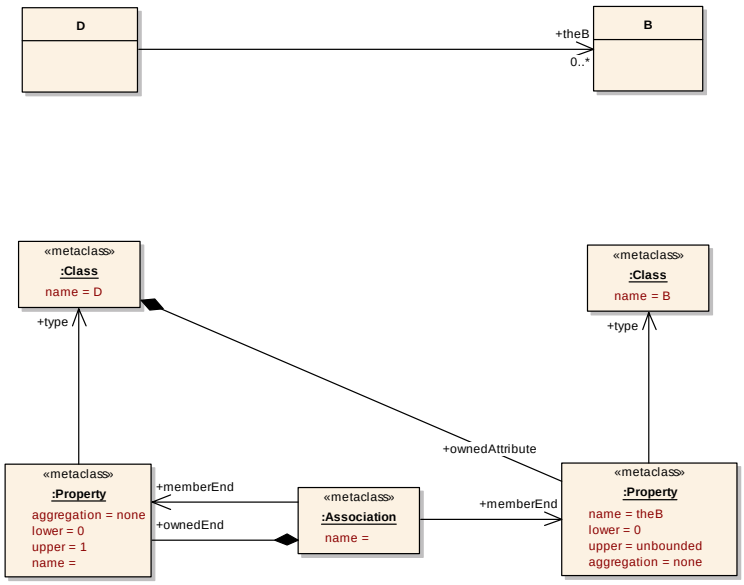


Figure 3-3: Representation of association (aggregation = none)

3.1.4 Representation of attribute (aggregation = none) [Tc0003Sc4]

Figure 3-4 depicts how an attribute with aggregation = none is represented by means of instances of the UML2.0 metamodel. The attribute ‘theB’ is represented as a property that is owned by class ‘D’.
Notes:

- A property with ‘aggregation = none’ is represented by a “*” in the UML2.0 graphical representation (attribute: theB: B*[0..*]).
- According to the AUTOSAR Template Profile and Modeling Guide [4] only attributes with aggregation=composite are allowed. However, the persistence rules for XML cover those attributes since they do not add complexity: For the persistence rules for XML attributes with aggregation=none (described in this section) are equivalent to associations with aggregation=none (described in section 3.1.3).

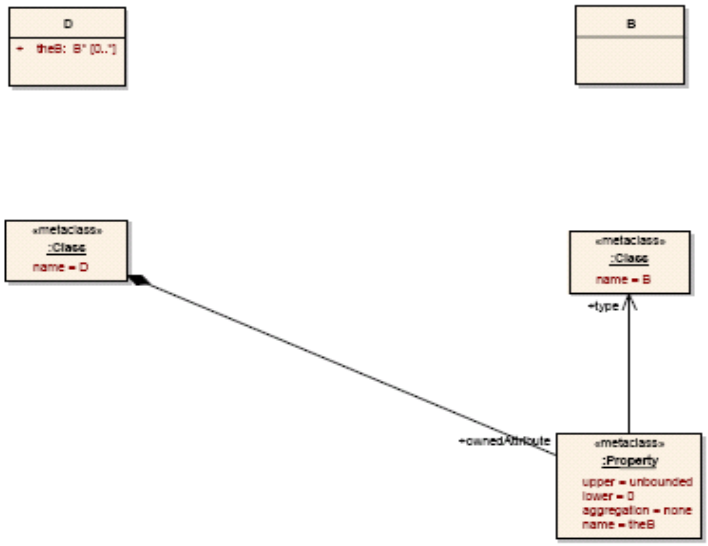


Figure 3-4: Representation of attribute (aggregation = none)

3.2 Notes on use of W3C XML schema

A W3C XML schema provides a means by which a validating XML parser can validate the syntax and some of the semantics of an XML description.
XML validation can determine e.g.

- whether required XML elements are available,
- whether additional XML attributes or XML elements that are not allowed are used,
- or whether some values fit to a given regular expression.

Although some checking can be done, it is impossible to rely solely on XML validation to verify that the represented model satisfies all of a model's semantic constraints.

The persistence rules described in this document make sure that for each class, attribute and association represented in the AUTOSAR metamodel a representation in the AUTOSAR XML schema exists. Additionally, they make sure that the mapping between the AUTOSAR metamodel and the AUTOSAR schema is unambiguous.
In other words:

- An instance of the AUTOSAR metamodel maps unambiguously to an AUTOSAR XML description and
- An AUTOSAR XML description that is valid with respect to the AUTOSAR XML schema maps unambiguously to an instance of the AUTOSAR metamodel.

This also holds for incomplete XML descriptions.
E.g.: The XML element `ATOMIC-SOFTWARE-COMPONENT-TYPE` always represents content that is described by the class AtomicSoftwareComponentType.

3.3 Handling inheritance

Inheritance in the AUTOSAR metamodel is mapped to XML schema by the following mechanisms:

- For each class in the AUTOSAR metamodel groups are created which contain the XML-elements and XML-attributes that represent the properties that are directly defined by the class.
- Additionally an xsd:complexType that represents the full content of concrete class is created. The structure of this xsd:complexType is defined by referencing to the group that define the properties of the class itself and the xsd:groups that define the properties of the superclasses. The groups representing the most general class (root of inheritance tree) is referenced first. The group representing the class itself is referenced last.

This concept allows for using polymorphism on XML level: The most general properties can always be found at the beginning of an XML-element. The more specific properties are appended at the end of a description.
Additionally properties that are directly defined by a class are grouped together.
(See section 3.7 for more details on the order of XML-elements and groups).

3.4 Generic approach

The AUTOSAR model persistence rules exceed the configuration possibilities of comparable approaches like XMI. This enables the generic reproduction of a wide range of existing XML formats such as MSR-SW [12] and XHTML [13].

3.5 XML element versus attribute

In accordance to the MSR-TR-CAP the persistence rules map all content related information to XML elements. This default rule can be overwritten by assigning the tagged value ‘xml.attribute=true’ to the respective property. If ‘xml.attribute=true’ then the property is translated to an XML-attribute. (See 4.1.2.1 for more details on this tagged value).

3.6 XML names [Tc0001]

All XML-elements, XML-attributes, XML-groups and XML-types used in the AUTOSAR XML schema are written in upper-case letters. In order to increase the readability of the XML names, hyphens are inserted in the XML names which separate the parts of the names.
This document refers to a name that is translated as described in this section as a **XML-name**.

Formally the following algorithm describes the translation of the UML names to XML names:

1. Split up the UML name from left to right into tokens. A new token starts whenever an uppercase letter or digit is found.
   E.g.: TestECUClass12ADC -> [Test, E, C, U, Class, 1, 2, A, D, C]

2. Iterate through the list, beginning from the last element and join adjacent single uppercase letters and join adjacent digits.
   E.g:
   [Test, E, C, U, Class, 1, 2, A, D, C] ->
   [Test, E, C, U, Class, 1, 2, A, DC] ->
   [Test, E, C, U, Class, 1, 2, ADC] ->
   [Test, E, C, U, Class, 12, ADC] ->
   [Test, E, CU, Class, 12, ADC] ->
   [Test, ECU, Class, 12, ADC]
3. Convert all tokens to uppercase:
   E.g: [TEST, ECU, CLASS, 12, ADC]
4. Append the tokens using a hyphen:
   E.g: TEST-ECU-CLASS-12-ADC

If the default mapping is not suitable, the XML name can be explicitly defined by specifying the tagged value ‘xml.name’ for the corresponding UML model element.

A plural XML-name is created by appending an “S” to the singular XML-name. It this rule is not suitable then the plural XML-name can be explicitly defined by specifying the tagged value ‘xml.namePlural’ for the corresponding UML model element.

Examples: XML names of elements, types, groups and attributes

The following table shows some examples of translations from metamodel names to names used in the XML schema:

Name in AUTOSAR metamodel	***XML name***
SystemConstraintTemplate	SYSTEM-CONSTRAINT-TEMPLATE
ECUResourceTemplate	ECU-RESOURCE-TEMPLATE
HardwarePowerMode	HARDWARE-POWER-MODE
Min	MIN
TestECUClass12ADC	TEST-ECU-CLASS-12-ADC
Uuid	UUID
testECU	TEST-ECU
MlData1	ML-DATA-1

3.7 Order of XML-elements [Tc0002]

In order to decrease the complexity and to improve the performance of tools that read AUTOSAR XML descriptions a predictable order of XML-elements is defined. Additionally the order of XML-elements improves the human readability of XML descriptions.

3.7.1 Order of xml elements

Properties owned by classes in the AUTOSAR metamodel are mapped to XML-elements. By default, the AUTOSAR XML schema defines a certain sequence on XML elements which follows an alphabetical ordering[1]. This default rule can be overwritten by using the tagged value 'xml.sequenceOffset'. The value can be all integers between -999 to 999. The default value of xml.sequenceOffset is 0.

If *offsetA* is the offset of `elementA` and *offsetB* is the offset of `elementB` then:

offsetA < *offsetB* => `elementA` is listed before `elementB`

Elements with the same offset are ordered alphabetically.

3.7.2 Order of xml elements of derived uml:properties

Chapter 3.7.1 described the order of the XML elements without considering inheritance. In case of inheritance not only the XML elements that are generated out of the properties that are directly defined by a class need to be considered. Additionally the XML-elements defined by the super-classes are relevant.

The XML elements that represent XML elements directly owned by a class are grouped together and ordered as described in section 3.7.1. The groups of XML elements are ordered as described by the pseudo code below:

```
// global variables
int index = 1;
Hashtable sequenceIndexTable = new Hashtable();

// method setSequenceIndex is invoked recursively
void setSequenceIndex(Class class) {
  List directBaseClasses = getDirectBaseClasses(class);

  // if class has baseclasses
  If ( !directBaseClasses.isEmpty() ) {
     // split up set of baseclasses into two groups
     List classesWithSupertypeIdentifiable =
          getClassesWithSupertypeIdentifiable(directBaseClasses);
     List classesWithoutSupertypeIdentifiable =
          getClassesWithoutSupertypeIdentifiable(directBaseClasses);

     // sort each group as defined above
     sort(classesWithSupertypeIdentifiable);
     sort(classesWithoutSupertypeIdentifiable);

     // for all classes with supertype identifiable do
     for (int i=0; i<classesWithSupertypeIdentifiable ; i++) {
        setSequenceIndex(classesWithSupertypeIdentifiable[i]);
     }
     // for all classes without supertype identifiable do
```

[1] The alphabetical ordering applies to the XML-names.

```
      for (int i=0; i<classesWithSupertypeIdentifiable ; i++) {
         setSequenceIndex(classesWithoutSupertypeIdentifiable[i]);
      }

   } // end if

   // if sequence index is not already set, assign a new one.
   If (sequenceIndexTable.contains(class)) {
      // the sequence is already set. This can result from diamond
      // inheritance
      return;
   } else {
      // the sequence index is not yet set
      sequenceIndexTable.put(class, index);
      index++;
   }
}
```

Figure 3-5 shows an **example** of the ordering of XML elements within the XML schema. The numbers next to class `Identifiable` indicate to sequenceOffset of the directly owned properties. The comments indicate the sequence of the groups of XML elements.

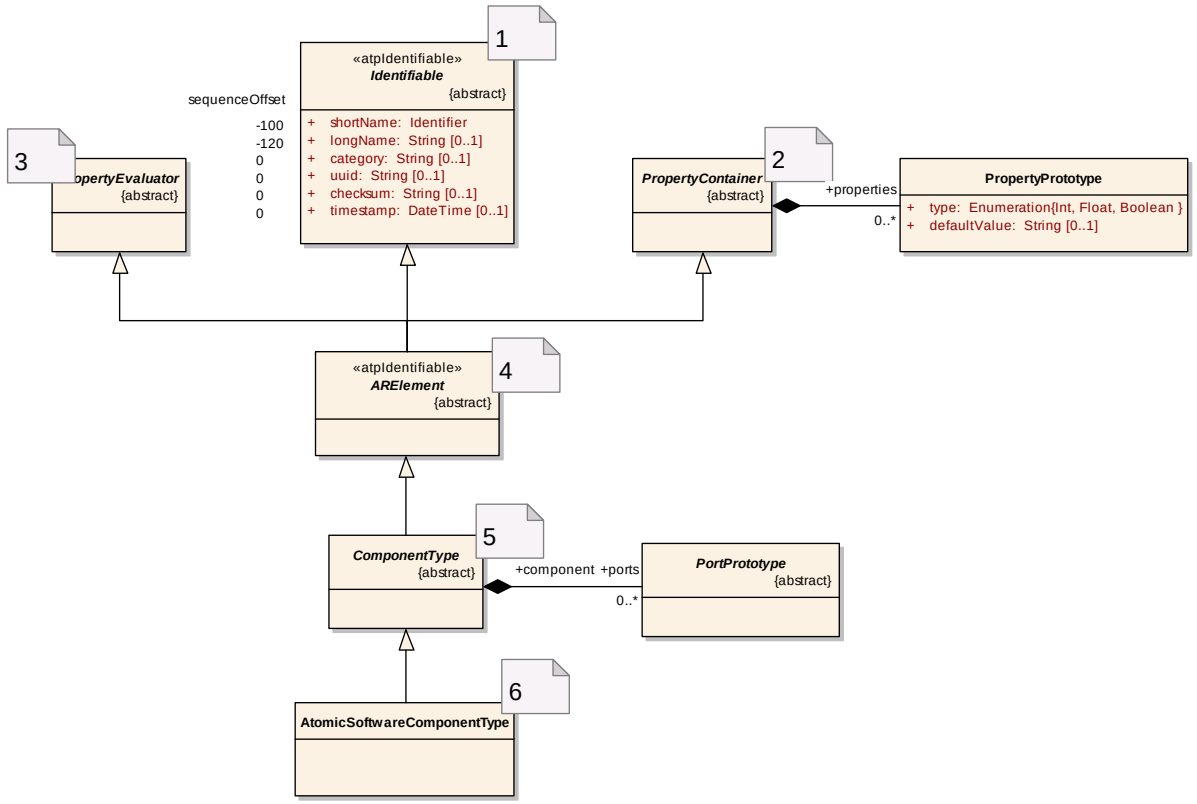


Figure 3-5: Order of XML elements

First the attributes from `Identifiable` are mapped to the XML schema. After that the properties from `PropertyContainer`, `PropertyEvaluator`, `ARElement`, `ComponentType` and `AtomicSoftwareComponentType` are mapped.

A xsd:group is created for all classes which directly own properties. In this example xsd:groups are created for `PropertyContainer`, `Identifiable` and `ComponentType`. The XML elements in each xsd:group are ordered as defined in section 3.7.1 (in this examples all properties are mapped to XML elements):

- xsd:group for `Identifiable`:

```
<xsd:group name="IDENTIFIABLE">
  <xsd:sequence>
    <xsd:element name="LONG-NAME" type="xsd:string" minOccurs="0"/>
    <xsd:element name="SHORT-NAME" type="AR:IDENTIFIER"/>
    <xsd:element name="CATEGORY" type="xsd:string" minOccurs="0"/>
    <xsd:element name="CHECKSUM" type=" xsd:string " minOccurs="0"/>
    <xsd:element name="TIMESTAMP " type=" xsd:string " minOccurs="0"/>
    <xsd:element name="UUID" type=" xsd:string " minOccurs="0"/>
  </xsd:sequence>
</xsd:group>
```

  LONG-NAME is listed first because it has the smallest sequenceOffset (-120). It is followed by SHORT-NAME (sequenceOffset=-100). All other properties have a sequenceOffset=0 and are sorted in alphabetical order.

- xsd:group for `PropertyContainer`:

```
<xsd:group name="PROPERTY-CONTAINER">
  <xsd:sequence>
    <xsd:element name="PROPERTIES" minOccurs="0" maxOccurs="unbounded" >
      …
    </xsd:element>
  </xsd:sequence>
</xsd:group>
```

- xsd:group for `ComponentType`:

```
<xsd:group name="COMPONENT-TYPE">
  <xsd:sequence>
    <xsd:element name="PORTS" minOccurs="0" maxOccurs="unbounded" >
      …
    </xsd:element>
  </xsd:sequence>
</xsd:group>
```

According to the rules for order of elements in case of inheritance these xsd:groups are composed together in the following order:

```
<xsd:complexType name="ATOMIC-SOFTWARE-COMPONENT-TYPE" abstract="false" mixed="false">
  <xsd:sequence>
    <xsd:group ref="AR:IDENTIFIABLE"/>
    <xsd:group ref="AR:PROPERTY-CONTAINER"/>
    <!—id3: element group for PROPERTY-EVALUATOR not generated -->
    <!—id4: element group for AR-ELEMENT not generated -->
    <xsd:group ref="AR:COMPONENT-TYPE"/>
    <!— id5: element group for ATOMIC-SOFTWARE-COMPONENT-TYPE not generated -->
  </xsd:sequence>
</xsd:complexType>
```

3.8 Linking [Tc0004]

References between metaclasses are represented through XML-elements suffixed by <….-REF> or <…-TREF>. Referenced XML-descriptions must define a <SHORT-NAME> which must be unique within its namespace.[2] Therefore a XML-description can be referenced by specifying an absolute path.

[1] [2] The namespace hierarchy is defined in the Meta-Model by composite association of classes derived from Identifiable. Each Identifiable is namespace of its directly or indirectly associated (composite association) classes.

3.8.1 Referencing via SHORT-NAME versus ID/ID-REF

An AUTOSAR XML description can be split up over several documents. For example there might be one file for all units, another document for all data-types, etc. The standard XML ID/ID-REF referencing mechanism is not suitable for cross document references: The semantics of ID/ID-REF requires the target of a reference to be available in the same document. Otherwise the XML-parser reports an error.
Additionally the IDs can only be defined globally. No namespace concept is supported.
The referencing mechanism via <SHORT-NAMES> doesn't require all referenced elements to be available in one document. Since this referencing mechanism bypasses the ID/ID-REF mechanism, unresolved references within one document don't result in invalid XML files.
Furthermore the <SHORT-NAME> referencing mechanism provides the concept of namespaces. Therefore the value of <SHORT-NAME> is not required to be globally unique and "speaking identifiers" can be used. When resolving <SHORT-NAME> references, all documents are logically merged into one single document. No explicit specification of the file-name in the references is required.
The following subsections give a more detailed description on the <SHORT-NAME> referencing mechanism and the merging of AUTOSAR XML documents.

3.8.2 References to elements via SHORT-NAME paths

References are represented by XML elements which describe the following information:

1. The absolute SHORT-NAME path of the referenced object.
2. The destination type of the reference.

3.8.2.1 Absolute SHORT-NAME path

SHORT-NAME paths are composed of sequences of <SHORT-NAME>s separated by "/". The following rules apply to SHORT-NAME paths used in AUTOSAR:

1. An absolute path is calculated by collecting the SHORT-NAMES of the model elements on the containment path from root element of the model to the referenced element.
2. Absolute paths begin with the character "/".
3. No relative paths are allowed.

3.8.2.2 Destination type

The destination type defines the type of the referenced object. The destination type improves the robustness of the XML descriptions: E.g.:

- A tool can find references which refer to objects of the wrong type.
- If the referenced object is not available the tool can indicate the correct type

3.8.2.3 Example

Figure 3-6 depicts a simple reference between class D and the abstract class B.

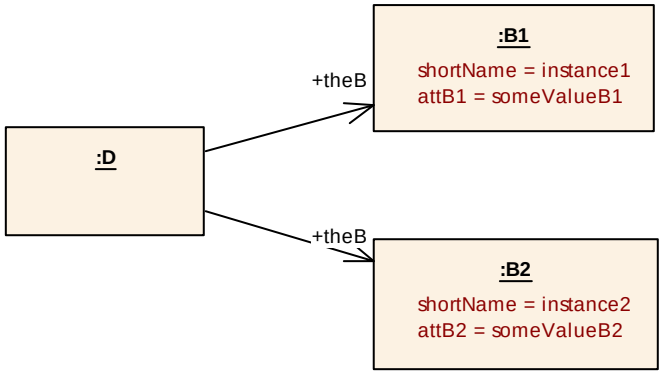


Figure 3-6: Example of reference (on M1)

The XML representation of these references is:

```
<D>
  <THE-B-REFS>
    <THE-B-REF DEST=”B-1”>
      /package1/package2/instance1
    </THE-B-REF>
    <THE-B-REF DEST=”B-2”>
      /package1/package2/instance2
    </THE-B-REF>
  </THE-B-REFS>
</D>
```

3.9 Transmitting incomplete Data

In order to allow the exchange of incomplete AUTOSAR descriptions, by default all XML elements are optional.
This default rule can be overwritten by marking a metamodel element with the tagged value ‘xml.enforceMinimumMultiplicity=true’. In This case the lower value of the multiplicity will be used as minimum occurrence of an XML element in the AUTOSAR XML schema.

3.10 Identification of XML schema version in XML descriptions

The version of the AUTOSAR release is encoded into the namespace of the AUTOSAR XML schema. The format of the namespace is defined by http://autosar.org/<version>. This allows for parallel use of different versions of AUTOSAR XML schema and AUTOSAR XML descriptions. E.g. the namespace of the AUTOSAR XML schema published with AUTOSAR release 2.1 is http://autosar.org/2.1.0. (The last part of the version number is reserved for bug fix releases.

3.11 Representation of primitive data types

In the AUTOSAR metamodel primitive datatypes are represented by uml:classes which are marked by the stereotype <<primitive>>. The tagged values xml.xsd.type and xml.mds.type define which xsd:simpleType represents the primitive uml: datatype. xml.mds.type selects a datatype that is defined within this document (see section 5.6). xml.xsd.type selects a datatype that is defined in the W3C XML schema specification.
The following table list the mapping between primitive uml:datatypes and xsd:primitiveTypes as they are defined in the AUTOSAR metamodel release 2.1:

uml:datatype	***xsd:simpleType***
String	xsd:string
Integer (and Int)	xsd:integer
Boolean	xsd:boolean
DateType	xsd:dateTime
Float	xsd:double
Identifier	AR:IDENTIFIER
UnlimitedNatural	xsd:nonNegativeInteger

4 Configuration of XML schema production [Tc0005]

In order to reduce the complexity of the mapping rules and to increase the transparency between metamodel classes, attributes and associations, and XML-elements the mapping rules do not directly operate on the AUTOSAR metamodel. Instead it is translated in two steps:

- **Step1: Configuration of XML schema production**.
  In a first step the AUTOSAR metamodel is translated to a simplified intermediate model[3]. Content relevant information of the AUTOSAR metamodel is mapped to classes, properties, enumerations, primitive datatypes and tagged values. If not otherwise mentioned all missing tagged values are set to the default values as described in section 4.1.1ff. Some more complex mappings are described in chapter 4.2.
- **Step 2: Schema production.**
  The schema production rules use the intermediate model as input. The rules are parameterized by the tagged values defined in step 1. See chapter 5 for more details on the Schema production rules.

4.1 Tailoring schema production

4.1.1 Overview

The tagged values that can be used for tailoring the mapping rules are described in the following table. The effect of the different tagged values is shown in the detailed description of the mapping rules in chapter 5.

Tag Name	*Value Type*	*Default Value*	*Applicable to*	*Description*
xml.name [Tc0005Sc1]	String	See chapter 3.6	Property, Class	Provides the name of a schema fragment (element, attribute, group, etc.) that represents the role or class. If not explicitly defined in the AUTOSAR metamodel, then this value is calculated as explained in chapter 3.6.
xml.namePlural [Tc0005Sc2]	String	See chapter 3.6	Property, Class	Provides the plural name of a schema fragment (element, attribute, group, etc.) that represents the role or class. If not explicitly defined in the AUTOSAR metamodel, then this value is calculated as explained in chapter 3.6.
xml.roleElement [Tc0010]	Boolean	See chapter 4.2.2	Property	If set to true, the xml.name of the role shows up as an XML-element. If not explicitly defined in the AUTOSAR metamodel, then the default rules as described in chapter 4.2 are applied.

[3]The intermediate model only uses concepts which are available in EMOF [17].

Tag Name	*Value Type*	*Default Value*	*Applicable to*	*Description*
xml.roleWrapperElement [Tc0010]	Boolean	See chapter 4.2.2	Property	If set to true, the xml.namePlural of the role shows up as an XML-element. This XML element is usually used to group several role elements or type elements. If not explicitly defined in the AUTOSAR metamodel, then the default rules as described in chapter 4.2 are applied.
xml.typeElement [Tc0010]	Boolean	See chapter 4.2.2	Property	If set to true, the xml.name of the type shows up as an XML-element. If not explicitly defined in the AUTOSAR metamodel, then the default rules as described in chapter 4.2 are applied.
xml.typeWrapperElement [Tc0010]	Boolean	false, see also chapter 4.2.2	Property	If set to true, the xml.namePlural of the type shows up as an XML-element. The type wrapper wraps several occurrences of the same type. If not explicitly defined in the AUTOSAR, then the default rules as described in chapter 4.2 are applied.
xml.attribute [Tc0005Sc3]	Boolean	False	Property	If true, serializes the property as an XML attribute. By default all properties are mapped to XML elements.
xml.enforceMaxMultiplicity [Tc0005Sc4]	Boolean	True	Property	If true, enforce maximum multiplicity; otherwise, it is “unbounded”. By default xml.enforceMaxMultiplicity is true.
xml.enforceMinMultiplicity [Tc0005Sc4]	Boolean	False	Property	If true, enforce minimum multiplicity; otherwise, it is “0.” In order to allow for transmitting partial information, the minimum multiplicity is not enforced by default.
xml.ordered [Tc0005Sc5]	Boolean	True	Class	If true, the order of XML elements representing the properties of a class is defined in a fixed order. If false, the order of XML elements representing the properties of a class is defined in arbitrary order. Additionally all XML-elements may occur several times. Please note that the tagged value ‘xml.ordered’ applies to the full XML representation of the class: All XML-elements are ordered regardless if they are inherited or not.

Tag Name	*Value Type*	*Default Value*	*Applicable to*	*Description*
xml.text [Tc0005Sc6]	Boolean	False	Class	If true, text is allowed between the XML elements representing the properties. By default no text is allowed between the properties. Please note that the tagged value 'xml.text' applies to the full XML representation of the class: Text may be written between all XML-elements, regardless if they are inherited or not.
xml.sequenceOffset [Tc0002]	Integer	0	Property, Generalization	The sequenceOffset is used to define the order of XML elements representing owned and derived properties. The range of sequenceOffset varies from -999 to 999. The elements with the smallest sequenceOffset are created first. Elements which have the same sequenceOffset are ordered alphabetically. If not explicitly defined in the AUTOSAR metamodel, then the xml.sequenceOffset is set to 0.
xml.globalElement [Tc0005Sc7]	Boolean	False	Class	If true, a global xsd:element is created for the tagged class. This xsd:element can be used as the root element of an instance of the schema. This tag needs to be explicitly defined in the AUTOSAR metamodel. Usually only the metaclass AUTOSAR is represented by a globally defined XML element.
xml.xsd.type [Tc0005Sc8]	String	Empty	Class with stereotype <<primitive>>	This tag identifies the xsd:simpleType which represents the primitive datatype in the metamodel. The value refers to a W3C XML schema datatype. The value of the tagged value shall not contain the namespace of the W3C schema. E.g.: Instead of 'xsd:string' please use 'string'.
xml.mds.type [Tc0005Sc8]	String	Empty	Class with stereotype <<primitive>>	This tag identifies the xsd:simpleType or xsd:complexType which represents the primitive datatype in the metamodel. The value refers to a datatype definition that is in chapter 5.6 of this document.
xml.nsURI [Tc0005Sc9]	String	http://autosar.org/<version>	Package, Class	By default all XML-elements are assigned to the namespace http://autosar.org/<version> (<version> refers to the AUTOSAR release number. E.g. http://autosar.org/2.1.0 for AUTOSAR release 2.1. The last part of the version number can be used to indicate bug fix release). If the namespace is applied to a package then it is implicitly applied to all owned classes and packages not defining their own namespace.

Tag Name	*Value Type*	*Default Value*	*Applicable to*	*Description*
xml.nsPrefix [Tc0005Sc9]	String	AR	Package, Class	By default all XML-elements are assigned to the namespace prefix “AR”. If this namespace prefix is applied to a package then it is implicitly applied to all owned classes and packages not defining their own namespace.
extensionPoint [Tc0006Sc5] [Tc0006Sc6]	Boolean	False	Class	If set to true, then the class is mapped as it would have subclasses. This allows for later adding subclasses without loosing compatibility to older XML descriptions.

4.1.2 Constraints on tags

Some tags are not allowed to be used in combination with other tags. These constraints are listed in the next two subchapters.

4.1.2.1 Constraints on tags applied to properties

Constraints on tags applied to properties	xml.name	xml.namePlural	xml.roleElement	xml.roleWrapperElement	xml.typeElement	xml.typeWrapperElement	xml.attribute	xml.enforceMaxMultiplicity	xml.enforceMinMultiplicity	xml.sequenceOffset
xml.name	/									
xml.namePlural		/								
xml.roleElement			/				o			
xml.roleWrapperElement				/			o			
xml.typeElement					/		o			
xml.typeWrapperElement						/	o			
xml.attribute			o	o	o	o	/	o		o
xml.enforceMaxMultiplicity							o	/		
xml.enforceMinMultiplicity									/	
xml.sequenceOffset							o			/

If the tagged value ‘xml.attribute’ is set to true, then a XML attribute is created for the respective property. The name of the XML-attribute is defined by the tagged value ‘xml.name’. If the lower value of the multiplicity of the property is bigger than 0 and ‘xml.enforceMinMultiplicity’ is set to true, then the ‘use’ of the XML attribute is set to ‘required’. The tagged values ‘xml.roleElement’, ‘xml.roleWrapperElement’,

'xml.typeElement', 'xml.typeWrapperElement', 'xml.enforceMaxMultiplicity' and 'xml.sequenceOffset' are ignored if the tagged value 'xml.attribute' is set to 'true'.
Please note, that the tagged value 'xml.attribute' is only allowed if the upper multiplicity of the property is 1 and the type of the property is an enumeration or a primitive datatype.

4.1.2.2 Constraints on tags applied to classes

Constraint on tags applied to classes	xml.name	xml.namePlural	xml.ordered	xml.text	xml.globalElement	xml.xsd.type	xml.mds.type
xml.name	/					o	o
xml.namePlural		/				o	o
xml.ordered			/			o	o
xml.text				/		o	o
xml.globalElement					/	o	o
xml.xsd.type	o	o	o	o	o	/	o
xml.mds.type	o	o	o	o	o	o	/

The tagged values 'xml.xsd.type' and 'xml.mds.type' are used to specify a predefined data type which is defined in the W3C XML schema specification or in chapter 5.6 of this document. If these tagged values are applied, then all other tagged values are ignored.

4.1.2.3 Constraints on values of xml.*Element tagged values

The following table depicts which combinations of values of the xml.*Element tags are allowed. The column usage defines that a combination is either preferred, alternative, "handle with care" or not allowed. The first two categories always lead to consistent, unambiguous XML schema. The "handle with care" category describes mapping rules which might lead to invalid XML schema. Those mapping rules are allowed in order to be able to support some MSR-TR-CAP concepts.

xml.roleWrapperElement	xml.roleElement	xml.typeWrapperElement	xml.typeElement	description	Usage (P=preferred, A=alternative, H=handle with care, N=not allowed)	used in standard
0	0	0	0	Handle with care: The resulting pattern will result in ambiguous XML schema	H	MSR
0	0	0	1	Handle with care: Role Information is missing - might lead to ambiguous XML schema if two roles have the same type	H	MSR
0	0	1	0	Not allowed: typeWrapperElement without typeElement	N	
0	0	1	1	Handle with care: Role Information is missing - might lead to ambiguous descriptions if two roles have the same type	H	MSR
0	1	0	0	Preferred for properties without inheritance and upper multplicity = 1, Handle with care if used with inheritance	P	XMI2.0, XMI2.1, MSR
0	1	0	1	Preferred for properties with inheritance and upper multplicity = 1.	P	XMI1.2
0	1	1	0	Not allowed: typeWrapperElement without typeElement	N	
0	1	1	1	Alternative solution for 0101 if the typeElements need to be wrapped by a typeWrapperElements.	A	MSR
1	0	0	0	Not allowed: roleWrapperElement without roleElement or typeElement	N	
1	0	0	1	Preferred for properties with upper multiplicity > 1	P	MSR
1	0	1	0	Not allowed: typeWrapperElement without typeElement	N	
1	0	1	1	Alternative mapping for (1001) if the typeElements need to be wrapped by a typeWrapperElements.	A	MSR
1	1	0	0	alternative for properties without inheritance and upper multiplicity > 1, handle with care if used with inheritance	A	MSR
1	1	0	1	Alternative solution for properties with inheritance and upper multiplicity > 1 (1001)	A	MSR
1	1	1	0	Not allowed: typeWrapperElement without typeElement	N	
1	1	1	1	Alternative solution for (1001)	A	MSR

4.2 Default configuration of XML schema production [Tc0006]

This chapter describes how the XML persistence rules are configured for mapping the AUTOSAR metamodel to the AUTOSAR XML schema. Tagged values that are already defined in the AUTOSAR metamodel are not overwritten: The configuration rules defined in this chapter add missing tagged values. If the resulting combination of tagged values is invalid, an error needs to be indicated. The fault needs to be resolved by editing the tagged values in the AUTOSAR metamodel.

4.2.1 Configuration of multiplicities

The tagged values ‘xml.enforceMinMultiplicity’ and ‘xml.enforceMaxMultiplicity’ are set to the default values (see chapter 4.1.1) if not explicitly defined otherwise in the AUTOSAR metamodel. Additionally the multiplicities of all properties are updated according to the following rules:

- If ‘xml.enforceMinMultiplicity = false’, then
  set ‘lower multiplicity’ of property to 0.
- If ‘xml.enforceMinMultiplicity = true’, then
  no changes on ‘lower multiplicity’ of property.

- If ‘xml.enforceMaxMultiplicity = false’, then
  set ‘upper multiplicity’ of property to unbounded.
- If ‘xml.enforceMaxMultiplicity = true’, then
  no changes on ‘upper multiplicity’ of property.

4.2.2 Mapping configuration for properties

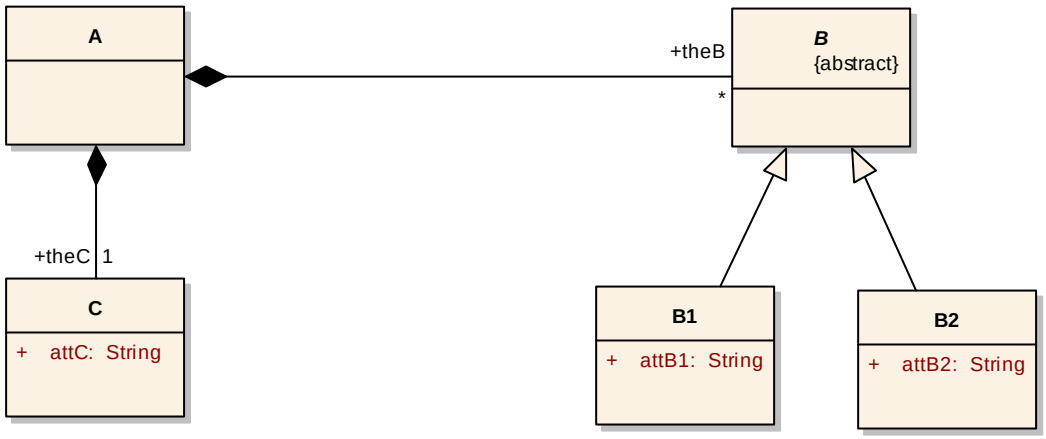


Figure 4-1: Example metamodel

Four cases are distinguished when configuring the mapping of properties in the AUTOSAR metamodel:

1. **Upper multiplicity of property = 1 and type of property has no subclasses (see property ‘theC’ in Figure 4-1): [Tc0006Sc1] [Tc0006Sc5]**
   - xml.roleWrapperElement = false
     Note: upper multiplicity of property = 1, no need for a wrapper
   - xml.roleElement = true
   - xml.typeWrapperElement = false
     Note: upper multiplicity of property = 1, no need for a wrapper
   - xml.typeElement = false
     Note: the type can uniquely be derived from metamodel

   Note: If the tagged value ‘extensionPoint’ is used for a class and set to true, then the class is mapped as it would have subclasses. This allows for later

adding subclasses without loosing backwards compatibility to older XML descriptions. In this case the "*Element" tagged values are set according to case 3.

2. **Upper multiplicity of property > 1 and type of property has no subclasses [Tc0006Sc2] [Tc0006Sc6]**
   - xml.roleWrapperElement = true
     Note: upper multiplicity of property > 1, according to MSR-TR-CAP wrapper required
   - xml.roleElement = false
     Note: property can be determined by the roleWrapperElement, no need for an additional roleElement
   - xml.typeWrapperElement = false
     Note: roleWrapperElement is true, no additional wrapper required
   - xml.typeElement = true
     Note: the content model of each type which occurs more than once needs to be encapsulated in an XML-element. Either the roleElement or the typeElement can be chosen. We chose the typeElement since the resulting schema allows for adding subclasses to the type of the property. (see also case 4)

3. **Upper multiplicity of property = 1 and type of property has subclasses [Tc0006Sc3]**
   - xml.roleWrapperElement = false
     Note: upper multiplicity of property = 1, no need for a wrapper
   - xml.roleElement = true
   - xml.typeWrapperElement = false
     Note: upper multiplicity of property = 1, no need for a wrapper
   - xml.typeElement = true
     Note: If no type information is given, it is not always possible to uniquely map an element in an XML description to an instance of the metamodel.

4. **Upper multiplicity of property > 1 and type of property has subclasses (see property 'theB' in Figure 4-1) [Tc0006Sc4]**
   - xml.roleWrapperElement = true
     Note: upper multiplicity of property > 1, according to MSR-TR-CAP wrapper required
   - xml.roleElement = false
     Note: property can be determined by the roleWrapperElement, no need for an additional roleElement
   - xml.typeWrapperElement = false
     Note: roleWrapperElement is true, no additional wrapper required
   - xml.typeElement = true
     Note: If no type information is given, it is not always possible to uniquely map an element in an XML description to an instance of the metamodel.

5. **Upper multiplicity of property > 1 and type of property is primitive or enum or association**
   - xml.roleWrapperElement = true
     Note: upper multiplicity of property > 1, according to MSR-TR-CAP wrapper required
   - xml.roleElement = true

   - xml.typeWrapperElement = false
     Note: roleWrapperElement is true, no additional wrapper required
   - xml.typeElement = false
     Note: the content model of each type which occurs more than once needs to be encapsulated in an XML-element. Either the roleElement or the typeElement can be chosen. For Primitives, we chose the roleElement since the MetaModel does not use subclassing for primitives.

Tagged values 'xml.*Element' that are already defined in the AUTOSAR metamodel are not overwritten. Therefore the mapping to XML can individually be configured if the default mappings are not sufficient.

4.2.3 Mapping configuration for references

In addition to the configuration defined in the previous section the following configuration is applied to references (association with aggregation = none).

4.2.3.1 References without stereotypes [Tc0004Sc1]

For references we basically distinguish the following two cases:

1. **Upper multiplicity of reference = 1**
   - Xml.RoleWrapperElement = false
   - xml.roleElememt = true
   - xml.typeWrapperElement = false
   - xml.typeElement = false

2. **Upper Multiplicity of reference > 1**
   - Xml.RoleWrapperElement = true
   - xml.roleElememt = true
   - xml.typeWrapperElement = false
   - xml.typeElement = false

Furthermore the 'xml.name' of properties representing the navigable association end of references is set to the default 'xml.name' appended by "-REF" (the default 'xml.name' is defined in chapter 3.6). The 'xml.namePlural' is set to the default 'xml.name' appended by "-REFS".

4.2.3.2 Instance references [Tc0004Sc2]

The AUTOSAR Template UML Profile and Modeling Guide[4] requires that all details of instance references are properly modeled in the AUTOSAR metamodel.

The following tagged values are applied:

- Composite reference between the source of the instance reference and the metaclass which contains the references to the context(s) and the target:
  - xml.name = xml-name suffixed by '-IREF'
  - xml.namePlural = xml-name suffixed by '-IREFS'
  - xml.typeWrapperElement = false
  - xml.typeElement = false
- instanceRef metaclass:
  - xml.name = xml-name suffixed by "-IREF". Additionally the "_" is replaced by "--" in order to guarantee the uniqueness of the xml name.
- reference from instanceRef metaclass to the target (the reference to the target is mandatory):
  - xml.enforceMinMultiplicity = true
  - xml.enforceMaxMultiplicity = true
  - xml.sequenceOffset = 9999 (the target is the last element in the list of references)
- references from instanceRef metaclass to contexts. For each context reference:
  - xml.enforceMinMultiplicity = false (references to the contexts may be added later)
  - xml.enforceMaxMultiplicity = true
  - xml.roleWrapperElement = false
  - xml.roleElement = true
  - xml.typeElement = false
  - xml.typeWrapperElement = false

xml.sequenceOffset = according to the sequence defined by the tagged value 'instanceRef.context'

Example

Figure 4-2 shows an example of a detailed representation of an instance reference in the AUTOSAR metamodel.

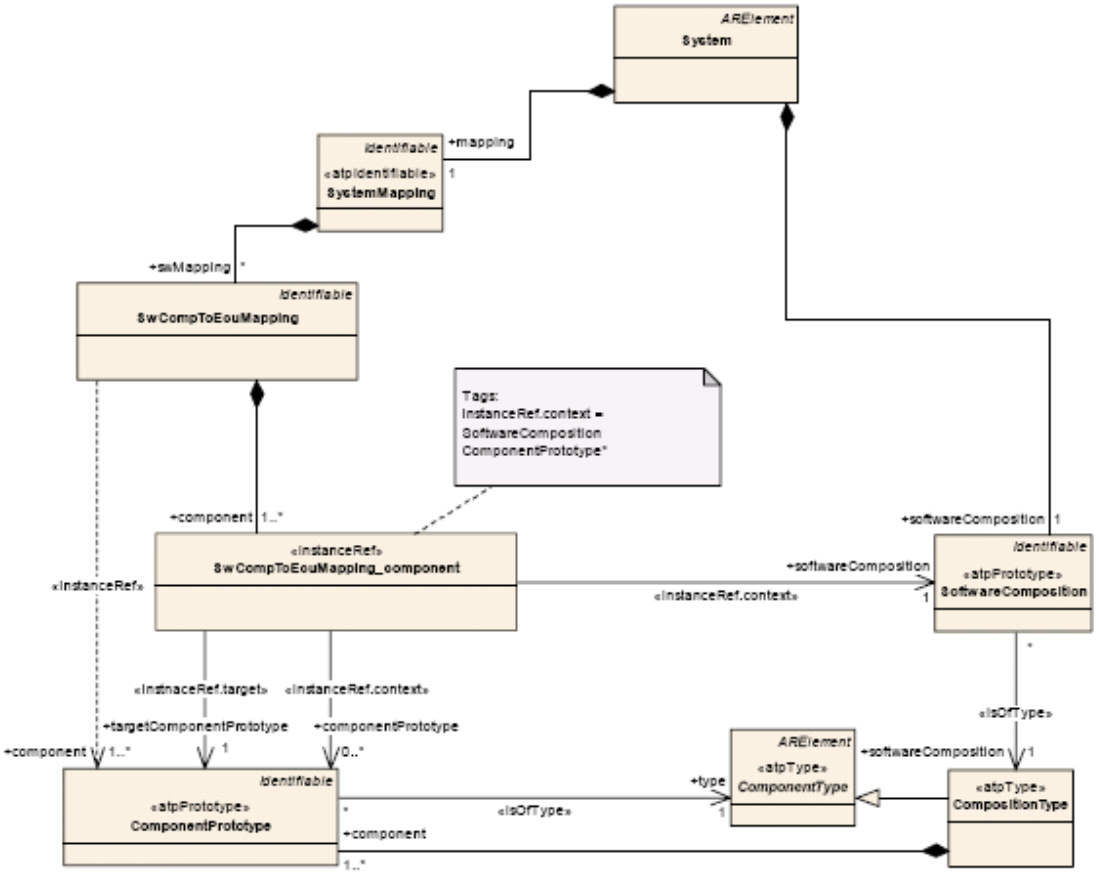


Figure 4-2: Example of an instanceRef association

The following XML snipped shows how this instance reference is represented in the XML schema:

```
<xsd:complexType name=”COMPONENT-IREF”>
 <xsd:sequence>
  <xsd:element name=”SOFTWARE-COMPOSITION-REF” minOccurs=”0” maxOccurs=”1”>
   <xsd:complexType>
    <xsd:complexContent>
     <xsd:restriction base=”AR:REF”/>
      <xsd:attribute name=”DEST” type=”AR:SOFTWARE-COMPOSITION—SUBTYPES-ENUM” use=”required”/>
    </xsd:complexContent>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name=”COMPONENT-PROTOTYPE-REF” minOccurs=”0” maxOccurs=”unbounded”>
   <xsd:complexType>
    <xsd:complexContent>
     <xsd:restriction base=”AR:REF”/>
      <xsd:attribute name=”DEST” type=”AR:COMPONENT-PROTOTYPE—SUBTYPES-ENUM” use=”required”/>
    </xsd:complexContent>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name=”TARGET-COMPONENT-PROTOTYPE-REF” minOccurs=”1” maxOccurs=”1”>
   <xsd:complexType>
    <xsd:complexContent>
     <xsd:restriction base=”AR:REF”/>
      <xsd:attribute name=”DEST” type=” AR:COMPONENT-PROTOTYPE—SUBTYPES-ENUM” use=”required”/>
    </xsd:complexContent>
   </xsd:complexType>
  </xsd:element>
```

```
  </xsd:sequence>
</xsd:complexType>
```

An example instaceRef looks like:

```
<…>
  <DEFECT-WHEEL-IREF>
     <IDENTIFIABLE-REF DEST=”CAR”>/vendor/someVehicle</IDENTIFIABLE-REF>
     <WHEEL-USAGE-REF DEST=”WHEEL-USAGE”>/vendor/someWheelUsage</WHEEL-USAGE-REF>
  </DEFECT-WHEEL-IREF>
</ …>
```

4.2.3.3 References with stereotype <<isOfType>> [Tc0004Sc2]

If the stereotype <<isOfType>> is applied to an association in the AUTOSAR metamodel then the tagged value ‘xml.name’ of the navigable association end is set to the default xml.name appended by “-TREF”. According to the “Template Profile and Modeling Guide” [4] the upper multiplicity of an association with stereotype <<isOfType>> is limited to 1. Therefore no multiplicity wrapper is required and no xml.namePlural needs to be defined.

4.2.4 Stereotypes applied to classes [Tc0011]

4.2.4.1 Stereotype <<atpMixed>> [Tc0011Sc1]

If the stereotype <<atpMixed>> is applied to a class in the AUTOSAR metamodel then the properties are represented by XML elements in arbitrary order and unbounded multiplicity. No wrappers are created for the properties. The following model transformation is applied if the stereotype <<atpMixed>> is applied:

- Modifications on the stereotyped metaclass:
  - xml.ordered=false
  - xml.text=false

- Modifications on the properties of the stereotyped metaclass:
  - upper multiplicity = unbounded
  - lower multiplicity = 0
  - xml.roleWrapperElement = false
  - xml.roleElement = true
  - xml.typeWrapperElement = false
  - xml.typeElement = true (if the type of the property has concrete subclasses), false (otherwise)

4.2.4.2 Stereotype <<atpMixedString>> [Tc0011Sc2]

If the stereotype <<atpMixedString>> is applied to a class in the AUTOSAR metamodel then the properties may be represented by XML elements in arbitrary order which may have text in-between. In this case the tagged value ‘xml.ordered’ is

set to false and the tagged value ‘xml.text’ is set to true. See chapter 4.1.1 for more details on the scope of the tagged value ‘xml.text’.

If the stereotype <<atpMixedString>> is applied to a class in the AUTOSAR metamodel then the properties are represented by XML elements in arbitrary order and unbounded multiplicity. No wrappers are created for the properties. Additionally the XML elements may have text in-between.The following model transformation is applied if the stereotype <<atpMixedString>> is applied:

- Modifications on the stereotyped metaclass:
  - xml.ordered=false
  - xml.text=true

- Modifications on the properties of the stereotyped metaclass:
  - upper multiplicity = unbounded
  - lower multiplicity = 0
  - xml.roleWrapperElement = false
  - xml.roleElement = true
  - xml.typeWrapperElement = false
  - xml.typeElement = true (if the type of the property has concrete subclasses), false (otherwise)

5 XML Schema production rules

The following sections describe the mapping rules for an automatic generation of the AUTOSAR XML schema out of the intermediate metamodel. Please note that in the intermediate metamodel all tagged values and multiplicities are set as defined in chapter 4.

Each rule is described by the following information:

- **Applies to:**
  The meta-metamodel (UML2.0) element the rule applies to
- **Precondition:**
  The rule can only be applied if the precondition evaluates to true
- **Target pattern**:
  The target pattern describes how the respective metamodel element is mapped to XML schema. Values that need to be read out of the AUTOSAR metamodel are denoted by script tags “<%” and “%>”:
  e.g.: <%=my variable %>.
  The following color code is used in the pattern definition:
  - **Black:** The part of the pattern which directly reflects the model element
  - **Yellow:** Information that is extracted or calculated from the AUTOSAR metamodel.
- **Description**:
  The description explains the target pattern and how it can be parameterized.
- **UML example, XML schema example and XML instance example:**
  These examples illustrate the application of the rule.

5.1 Create model representation [Tc0007]

Figure 5-1 depicts how a model is mapped to a XML schema. The header of the schema is created first, followed by XML representations for each class. After that the predefined data types and the footer of the schema are created.

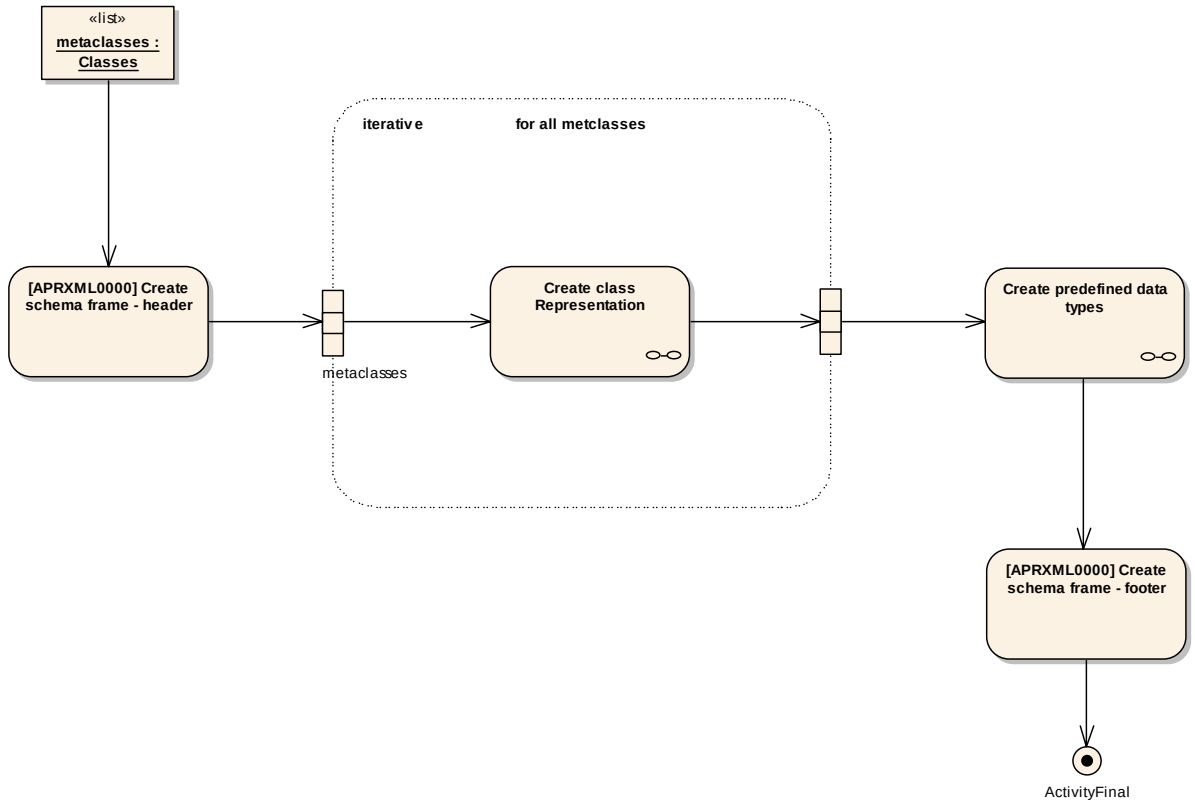


Figure 5-1: Model representation

5.1.1 [APRXML0000] Create xsd:schema

Applies to	Package
Precondition	n/a
Target pattern	<xsd:schema xmlns:<%=xmlNsPrefix%>="<%=xmlNsUri%>" xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="<%=xmlNsUri%>" elementFormDefault="qualified" attributeFormDefault="unqualified"> <% for all classes { %> <!—call class representation--> <% } %> <% <!—call rules APRXML0020 and APRXML0021 (predefined schema types)--> %> </xsd:schema>
Description	This rule creates the header and footer of the XML schema. By default xmlNsPrefix is set to “AR” and the xmlNsUri is set to “http://www.autosar.org/<version>”. The body of the XML schema is composed by representations for each class (including their properties) and a set of predefined data types.
UML example	n/a
XML schema	<xsd:schema xmlns:AR="http://autosar.org/2.1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema"

example	targetNamespace="http://autosar.org/2.1.0" elementFormDefault="qualified" attributeFormDefault="unqualified"> …. </xsd:schema>
XML instance example	<… xmlns="http://autosar.org/2.1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://autosar.org/2.1.0 autosar.xsd">…. </…>

5.2 Create class representation [Tc0008]

Figure 5-2 depicts XML schema fragments created for each class:

- If the stereotype <<enumeration>> is applied, then the class is mapped to an xsd:enumeration [APRXML0007] [Tc0008Sc5].
- If the stereotype <<primitive>> is applied, then the datatype denoted by the tagged value 'xml.mds.type' or 'xml.xsd.type' are used to represent the class within the schema [APRXML0006] [Tc0008Sc6].
- Otherwise:
  - If the class owns properties with 'xml.attribute=true' then an xsd:attributeGroup is created [APRXML0002] [Tc0008Sc4].
  - Additionally if the class owns properties with 'xml.attribute=false' then an xsd:group is created [APRXML0001] [Tc0008Sc3].
  - Additionally if the class is not abstract then an xsd:complexType is created [APRXML0003]. If the tagged value 'xml.globalElement' is set to true, then a global XML element declaration is created. [Tc0008Sc1]
  - Additionally if the uml:class is referenced then an xsd:simpleType that represents the lists possible concrete instances of the uml:class [APRXML00025].

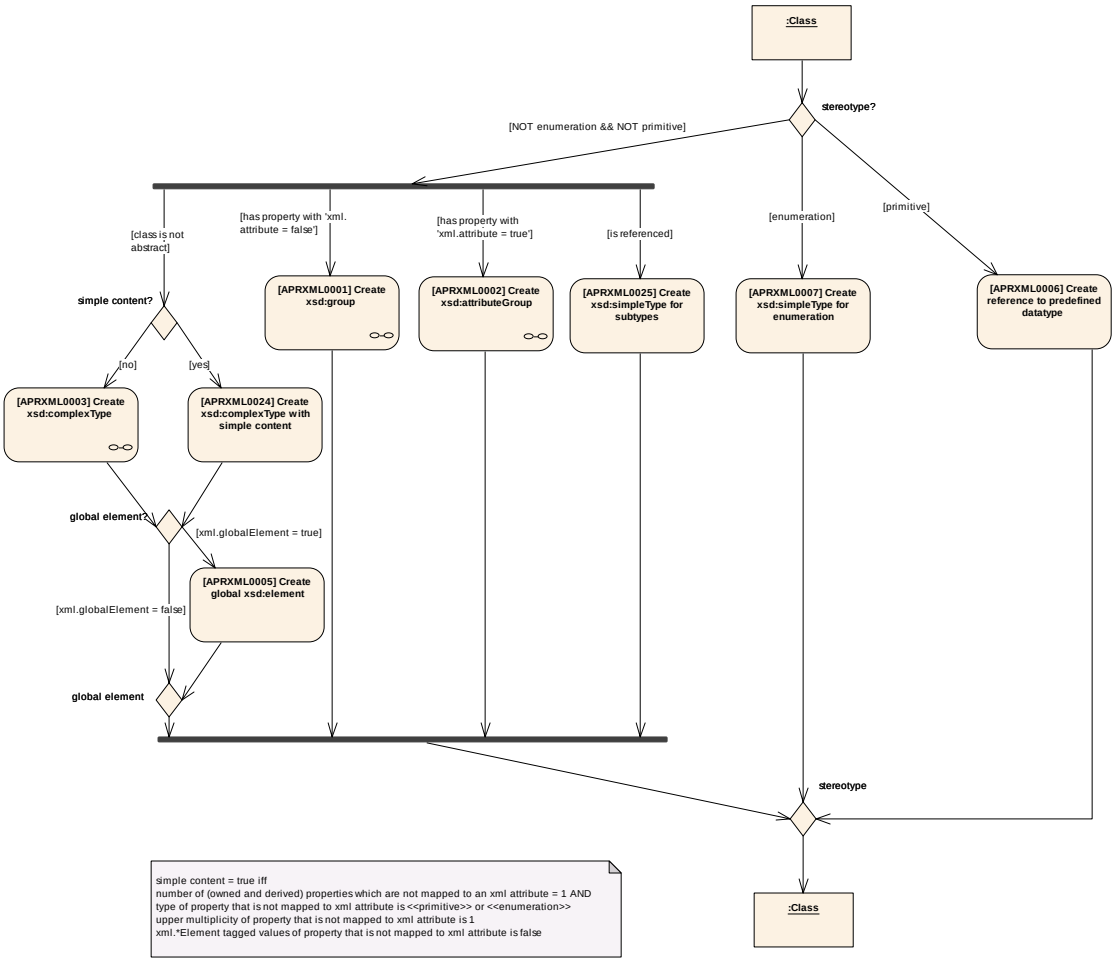
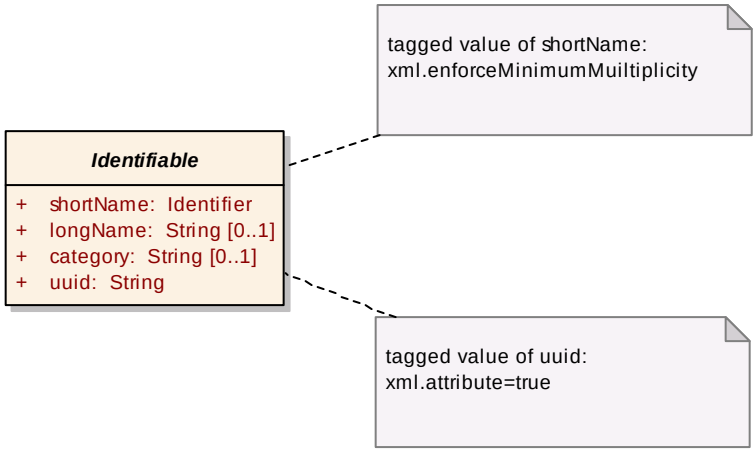


Figure 5-2: Class representation

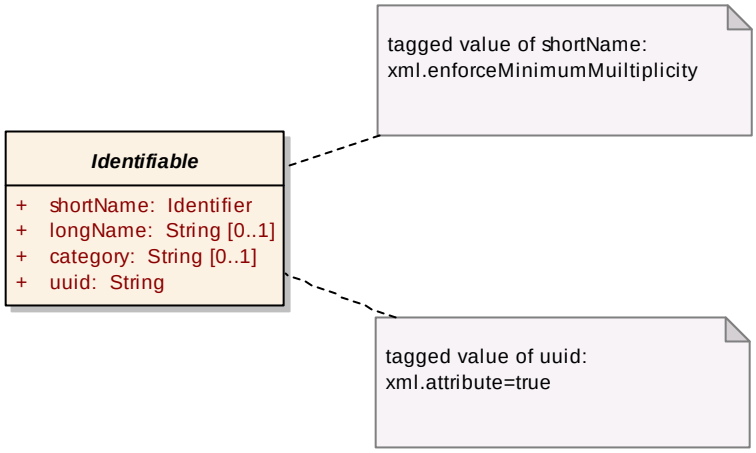
5.2.1 [APRXML0001] Create xsd:group [Tc0008Sc3]

Applies to	Class
Precondition	Exists properties with “xml.attribute=false”
Target pattern	<xsd:group name=”<%=xmlName%>”> <xsd:sequence> for all properties { %> <!—call rule APRXML0008 | APRXML0009 | APRXML0023 | APRXML0022 | APRXML0010 | APRXML0011 | APRXML0012 | APRXML0013 | APRXML0014 | APRXML0015 | APRXML0016 --> <% } // end for <xsd:sequence>

	</xsd:group>
Description	If the class owns at least one property with '**xml.attribute=false**', then a xsd:group is created. The name of the xsd:group maps to the XML-name of the class. The XML elements nested in the xsd:sequence are ordered as defined in section 3.7.1. The XML-elements representing the properties are created by rules APRXML0008, APRXML0009, APRXML0023, APRXML0022, APRXML0010, APRXML0011, APRXML0012, APRXML0013, APRXML0014, APRXML0015, APRXML0016
UML example	
XML schema example	<xsd:group name="IDENTIFIABLE"> <xsd:sequence> <!— property representations created by rules APRXML0008, APRXML0009, APRXML0023, APRXML0022, APRXML0010, APRXML0011, APRXML0012, APRXML0013, APRXML0014, APRXML0015, APRXML0016 --> <xsd:element name=”SHORT-NAME” type=”AR:IDENTIFIER” minOccurs=”1” maxOccurs=”1”> <xsd:element name=”LONG-NAME” type=”xsd:string” minOccurs=”0” maxOccurs=”1”> <xsd:element name=”CATEGORY” type=”xsd:string” minOccurs=”0” maxOccurs=”1”> <!— end property representations created by rules --> </xsd:sequence> </xsd:group>
XML instance example	<…> <SHORT-NAME>theShortName</SHORT-NAME> <LONG-NAME>theLongtName</LONG-NAME> <CATEGORY>theCategory</ CATEGORY > </…>

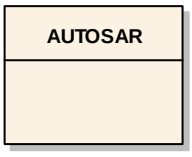
5.2.2 [APRXML0002] Create xsd:attributeGroup [Tc0008Sc4]

Applies to	Class
Precondition	Exists properties with “xml.attribute=true”
Target pattern	<xsd:attributeGroup name=”<%=xmlName%>”> <% for all properties represented as XML attributes { %> <!— call rule APRXML0019 --> <% } %} </xsd:attributeGroup>

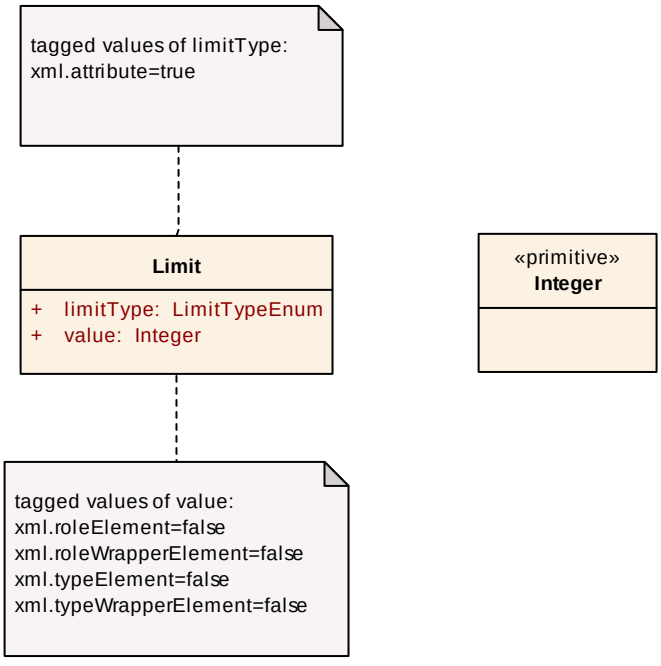
Description	If at least one property is marked by the tagged value 'xml.attribute=true', then a xsd:attributeGroup is created. The name of the xsd:attributeGroup is defined by the XML-name of the class which owns the property.
UML example	
XML schema example	<xsd:attributeGroup name="IDENTIFIABLE"> <!—attributes defined by rule APRXML0019 --> </xsd:attributeGroup>
XML instance example	n/a

5.2.3 [APRXML0003] Create xsd:complexType [Tc0008Sc1]

Applies to	Class
Precondition	isAbstract=false

Target pattern	<xsd:complexType name="<%=xmlName%>" mixed="<%=xmlText%>"> <% if (ordered) { %> <xsd:sequence> <% } else { %> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <% } <% for (myclass in {class and all baseclasses}) { %> <xsd:group ref="<%=baseclassNsPrefix%>:<%=myclassXmlName%>"/> <% } // for (class and all baseclasses) %> <% if (ordered) { %> </xsd:sequence> <% } else { %> </xsd:choice> <% } <% for (class and all baseclasses) { %> <xsd:attributeGroup ref="<%=baseclassNsPrefix%>:<%=baseclassXmlName%>"/> <% } // for (class and all baseclasses) %> </xsd:complexType>
Description	If the class is not abstract then a xsd:complexType is created. The name of the xsd:complexType is defined by the XML-name of the class. The created xsd:complexType doesn't directly define XML representations of properties. Instead it refers to the xsd:groups and xsd:attributeGroups which have been created for the class and all superclasses (see rule APRXML0001 and APRXML0002). The groups are ordered as defined in section 3.7.2. If the tagged value '**xml.ordered=true**' is set then the xsd:groups are listed in a xsd:sequence. Otherwise they are listed within a xsd:choice. If '**xml.text=true**' then the attribute 'mixed' of the xsd:complexType is set to 'true'.
UML example	
XML schema example	<xsd:complexType name="AUTOSAR"> <xsd:sequence> <xsd:group ref="AR:AUTOSAR"/> </xsd:sequence> <xsd:attributeGroup ref="AR:AUTOSAR"/> </xsd:complexType>
XML instance example	n/a

5.2.4 [APRXML0024] Create xsd:complexType with simple content

Applies to	Class
Precondition	isAbstract=false AND number of (owned and derived) properties which are not mapped to an xml attribute = 1 AND type of property that is not mapped to xml attribute is <<primitive>> or <<enumeration>> AND upper multiplicity of property that is not mapped to xml attribute is 1 AND xml.*Element tagged values of property that is not mapped to xml attribute is false
Target pattern	<xsd:complexType name="<%=xmlName%>" mixed="<%=xmlText%>"> <xsd:simpleContent> <xsd:extension base="<%=propertyTypeNsPrefix%>:<%=propertyTypeXmlName%>"> <% for (class and all baseclasses) { %> <xsd:attributeGroup ref="<%=baseclassNsPrefix%>:<%=baseclassXmlName%>"/> <% } // for (class and all baseclasses) %> </xsd:extension> </xsd:simpleContent> </xsd:complexType>
Description	If the class is not abstract and it contains exactly one (derived or owned) property that is not mapped to an xml.attribute and this property is not represented by any XML elements (tagged values xml.*Element=false) then a xsd:complexType with simpleContent is generated. The simpleContent contains the data of the property which is not mapped to an xml.attribute.
UML example	

XML schema example	<xsd:complexType name="LIMIT"> <xsd:simpleContent> <xsd:extension base="AR:INTEGER"> <xsd:attributeGroup ref="AR:LIMIT"/> </xsd:extension> </xsd:simpleContent> </xsd:complexType>
XML instance example	<... LIMIT-TYPE="CLOSED"> 1000 </...>

5.2.5 [APRXML0005] Create global xsd:element [Tc0008Sc2]

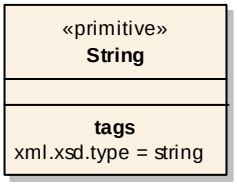
Applies to	Class
Precondition	'xml.globalElement=true
Target pattern	<xsd:element name="<%=typeXmlName%>" type="<%typeXmlNsPrefix%>:<%=typeXmlName%>"/>
Description	If the class is marked by the tagged value 'xml.globalElement=true' then a global xsd:element is created. The name of the xsd:element is defined by the XML-name of the class and the type is defined by the xsd:complexType which was defined by APRXML0003. The namespace prefix is defined by the tagged value 'xml.nsPrefix'.
UML example	
XML schema example	<xsd:element name="AUTOSAR" type="AR:AUTOSAR"/>
XML instance example	<AUTOSAR> … </AUTOSAR>

5.2.6 [APRXML0025] Create enumeration of subtypes

Applies to	Class
Precondition	Class is referenced
Target pattern	<xsd:simpleType name="<%=xmlName%>--SUBTYPES-ENUM"> <xsd:restriction base="xsd:string"> <% for not abstract type and all subtypes { %> <xsd:enumeration value="<%=typeXmlName%>"/> <%} // for not abstract type and all subtypes %> </xsd:restriction> </xsd:simpleType>

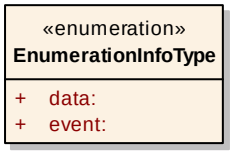
Description	Creates an enumeration which represents the XML names of the class (if not abstract) and all subtypes (if not abstract). This enumeration is required for describing potential destination types of references.
UML example	See Figure 5-7.
XML schema example	<xsd:element name="B—SUBTYPES-ENUM"> <xsd:restriction base="xsd:string"> <xsd:enumeration value="B-1"/> <xsd:enumeration value="B-2"/> </restriction> </xsd:element>
XML instance example	<THE-B-REF DEST="B-1">/shortname</THE-B-REF>

5.2.7 [APRXML0006] Create reference to XML predefined datatype [Tc0005Sc8] [Tc0008Sc6]

Applies to	class with stereotype <<primitive>>
Precondition	tagged value 'xml.xsd.type=…' or 'xml.mds.type=…' defined
Target pattern	…. type="<%=typeXmlNsPrefix%>:<%=xmlXsdType%>" ….
Description	Each class with the stereotype <<primitive>> is represented by the xsd:simpleType that is defined by the tagged value 'xml.xsd.type' or 'xml.mds.type'. If *xml.xsd.type* is used, then the type is defined in the W3C xml schema and the *typeXmlNsPrefix* corresponds to "xsd". If *xml.mds.type* is used, then the type is defined in the namespace of the generated XML schema ("AR"). These types are defined by rules APRXML0020 and APRXML0021.
UML example	
XML schema example	The predefined W3C XML schema datatype string is used to represent the primitive class String.
XML instance example	<…. type="xsd:string" ….>

5.2.8 [APRXML0007] Create xsd:simpleType for enumeration [Tc0008Sc5]

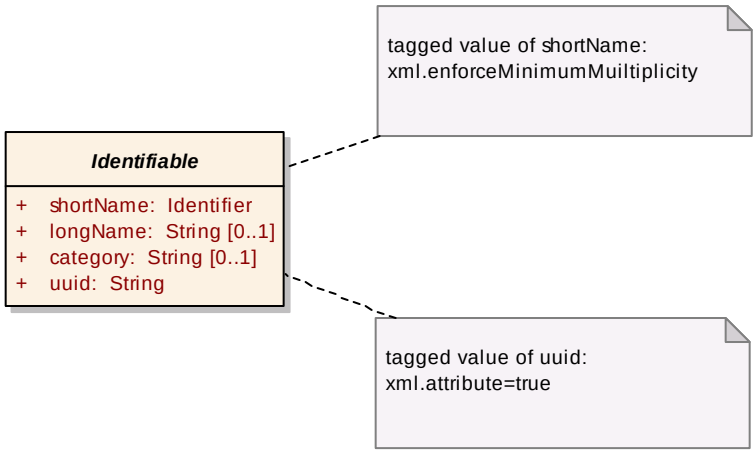
Applies to	class with stereotype <<enumeration>>

Precondition	n/a
Target pattern	<xsd:simpleType name="<%=xmlName%>"> <xsd:restriction base="xsd:string"> <% for all attributes { %> <xsd:enumeration value="<%=attributeXmlName%>"/> <%} // end for all attributes %> </restriction> </xsd:simpleType>
Description	A xsd:simpleType is created for each class which is marked by the stereotype <<enumeration>>. The name of the xsd:element maps to the XML-name of the class. The xsd:simpleType defines an enumeration. The enumeration literals are defined by the property names of the class.
UML example	
XML schema example	<xsd:simpleType name="ENUMERATION-INFO-TYPE"> <xsd:restriction base="xsd:string"> <xsd:enumeration value="data"/> <xsd:enumeration value="event"/> </xsd:restriction> </xsd:simpleType>
XML instance example	"data"

5.3 Create composite property representation (mapping to XML attributes) [Tc0009]

5.3.1 [APRXML0019] Create xsd:attribute

Applies to	Property
Precondition	xml.attribute=true upper multiplicity of property = 1
Target pattern	<xsd:attribute name="<%=xmlName%>" type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>" <% if lowerMultiplicity > 0 { %> use="required" <% } else { %> use="optional" <% } %> />
Description	An xsd:attribute is created for each property with the tagged value '**xml.attribute=true**'. The name of the xsd:attribute is defined by the XML name of the represented property. If the lower multiplicity of the property is bigger than 0 then the use of the attribute is required, otherwise it is optional.

UML example	
XML schema example	<xsd:attribute name="UUID" type="xsd:string" use="optional"/>
XML instance example	<…. UUID="12343-23342-12345-2333"/>

5.4 Create composite property representation (mapping to XML elements) [Tc0010]

Composite properties are properties with 'aggregation=composite'. If the tagged value 'xml.attribute=false' (default), then those properties are mapped to XML-elements. Depending on the values of the tagged values 'xml.roleWrapperElement', 'xml.roleElement', 'xml.typeWrapperElement' and 'xml.typeElement' one of the following rules is chosen. All rules that map composite properties to XML elements are called 'Composite Property Representation' extended by a number denoting the settings of the aforementioned tagged values. The first digit reflects the value of 'xml.roleWrapperElement', the second digit reflects the value of 'xml.roleElement', the third digit reflects the value of 'xml.typeWrapperElement' and the last digit reflects the value of 'xml.typeElement'.

Figure 5-3 illustrates how the rules are chosen based on the tagged values.

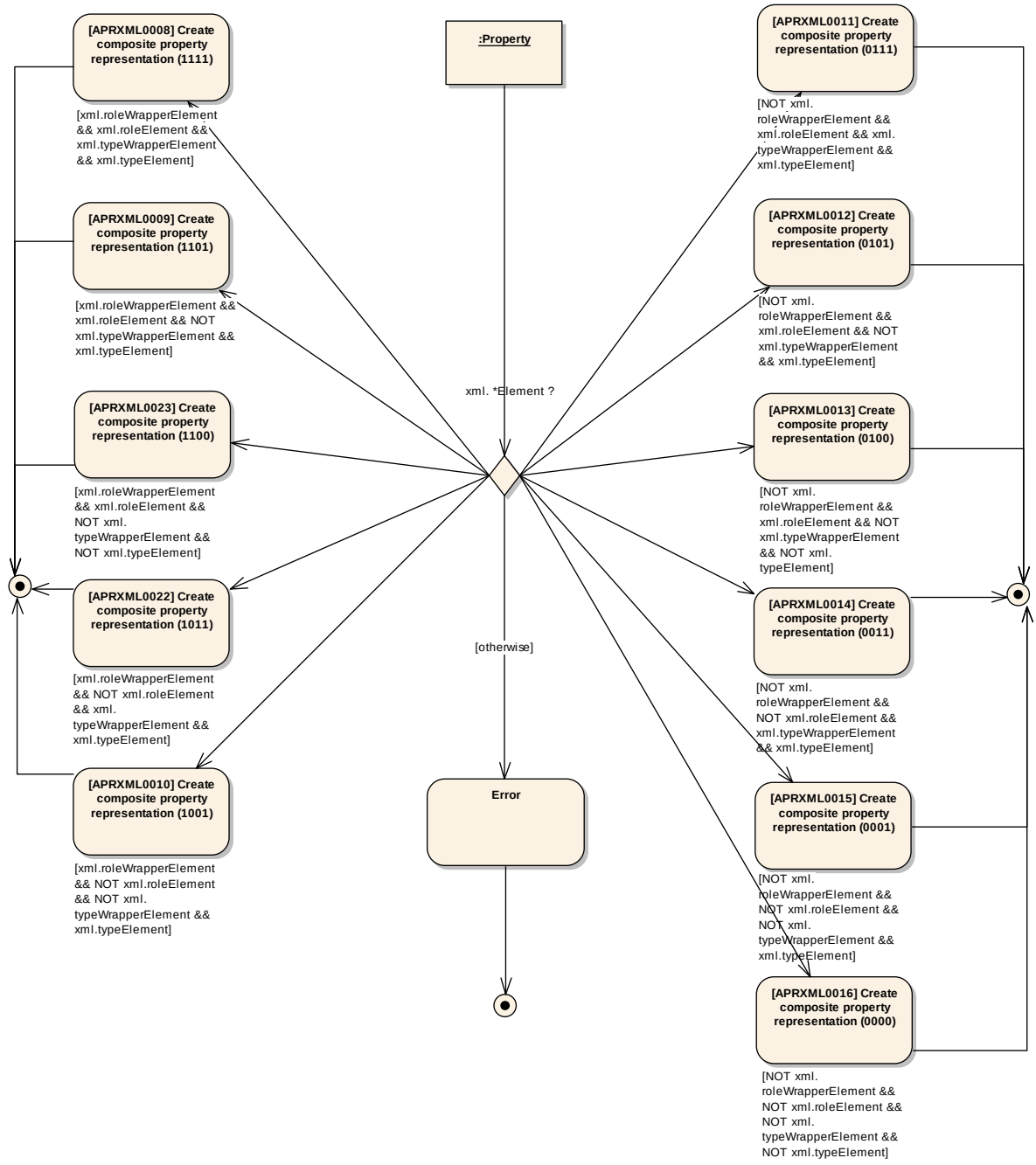


Figure 5-3: Property representation (aggregation = composite)

Figure 5-4 shows an example metamodel that is used as an example for the following mapping rules. The class ‘A’ owns a property called ‘theB’ which is of type ‘B’. The multiplicity of this property is 0..*. Additionally the class ‘A’ owns a property called ‘theC’ which is of type ‘C’. The upper multiplicity of ‘theC’ is 1 and the lower multiplicity is 1.

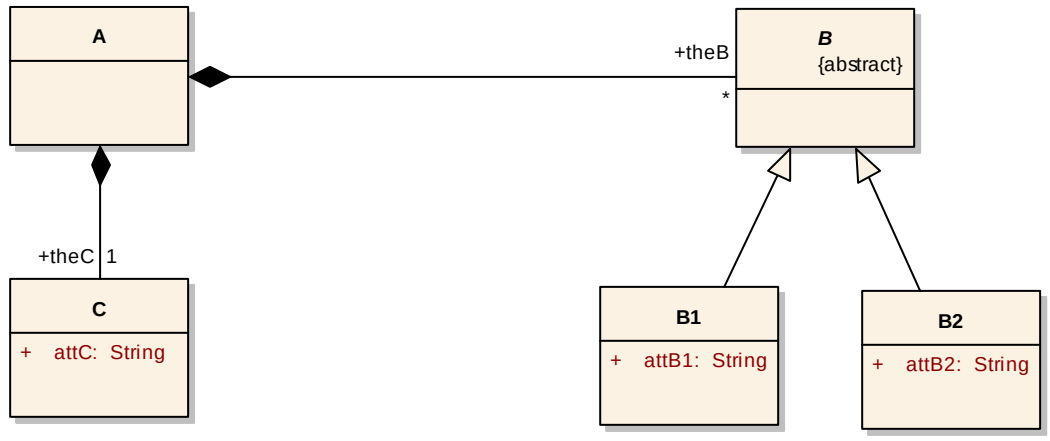


Figure 5-4: Example metamodel of composite property

Figure 5-5 shows an example instance of the metamodel shown in Figure 5-4. This instance is used as a basis for the XML instance examples of the following rules.

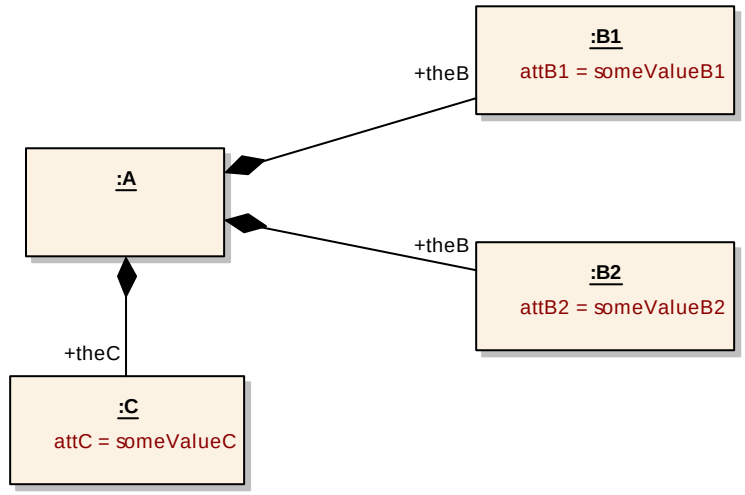


Figure 5-5: Example instance of composite property

5.4.1 [APRXML0008] Create composite property representation (1111) [Tc0010Sc1111]

Applies to	Property
Precondition	xml.roleWrapperElement == true && xml.roleElement == true && xml.typeWrapperElement == true && xml.typeElement == true

Target pattern	<%
	<xsd:element name="<%=roleXmlNamePlural%>"
	minOccurs="<%if (lowerMultiplicity.equals("0")) {%>0<%} else {%>1<%}%>"
	maxOccurs="1">
	<xsd:complexType>
	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>">
	<xsd:element name="<%=roleXmlName%>">
	<xsd:complexType>
	<xsd:all minOccurs="<%if (lowerMultiplicity.equals("0")) {%>0<%} else {%>1<%}%>">
	<%
	for all types {
	%>
	<xsd:element name="<%=typeXmlNamePlural%>" minOccurs="0" maxOccurs="1">
	<xsd:complexType>
	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>">
	<xsd:element name="<%=typeXmlName%>"
	type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>"/>
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>
	<%
	} // end for all types
	%>
	</xsd:all>
	</xsd:complexType>
	</xsd:element>
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>

Description	A set of XML elements is created that represent the property: • A role wrapper XML element with maxOccurs=1. The name of the XML element is defined by the xml.namePlural of the property. • Role XML elements are nested within the role wrapper element. The name of these XML-elements is defined by the xml.name of the property. The minimum occurrence of these elements is the lower multiplicity of the property; the maximum occurrence is the upper multiplicity. • For each type and subtype of the property a type wrapper XML element is created. The name of the XML element is defined by the xml.namePlural of the type of the property. • Nested in these type wrappers only elements representing the same types as the type wrapper are allowed.
UML example	See above

XML schema example

```
<xsd:element name="THE-BS" minOccurs="0" maxOccurs="1">
 <xsd:complexType>
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
   <xsd:element name="THE-B" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:all minOccurs="0">
      <xsd:element name="B1S" minOccurs="0" maxOccurs="1">
       <xsd:complexType>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
         <xsd:element name="B1" type="AR:B1"/>
        </xsd:choice>
       </xsd:complexType>
      </xsd:element>
      <xsd:element name="B2S" minOccurs="0" maxOccurs="1">
       <xsd:complexType>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
         <xsd:element name="B2" type="AR:B2"/>
        </xsd:choice>
       </xsd:complexType>
      </xsd:element>
     </xsd:all>
    </xsd:complexType>
   </xsd:element>
  </xsd:choice>
 </xsd:complexType>
</xsd:element>
<xsd:element name="THE-CS" minOccurs="0" maxOccurs="1">
 <xsd:complexType>
  <xsd:choice minOccurs="0" maxOccurs="1">
   <xsd:element name="THE-C" minOccurs="0" maxOccurs="1">
    <xsd:complexType>
     <xsd:all minOccurs="0">
      <xsd:element name="CS" minOccurs="0" maxOccurs="1">
       <xsd:complexType>
        <xsd:choice minOccurs="0" maxOccurs="1">
         <xsd:element name="C" type="AR:C"/>
        </xsd:choice>
       </xsd:complexType>
      </xsd:element>
     </xsd:all>
    </xsd:complexType>
   </xsd:element>
  </xsd:choice>
 </xsd:complexType>
</xsd:element>
```

XML instance example

```
<…>

 <THE-BS>
  <THE-B>
   <B1S>
    <B1>
     <ATT-B1>someValueB1</ATT-B1>
    </B1>
   </B1S>
   <B2S>
    <B2>
     <ATT-B2>someValueB2</ATT-B2>
    </B2>
   </B2S>
  </THE-B>
 </THE-BS>


 <THE-CS>
  <THE-C>
   <CS>
    <C>
     <ATT-C>someValueC</ATT-C>
    </C>
   </CS>
  </THE-C>
 </THE-CS>

</…>
```

5.4.2 [APRXML0009] Create composite property representation (1101) [Tc0010Sc1101]

Applies to	Property
Precondition	xml.roleWrapperElement == true && xml.roleElement == true && xml.typeWrapperElement == false && xml.typeElement == true

Target pattern	<xsd:element name="<%=roleXmlNamePlural%>"
	minOccurs="<%if (lowerMultiplicity.equals("0")) {%>0<%} else {%>1<%}%>"
	maxOccurs="1">
	<xsd:complexType>
	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>">
	<xsd:element name="<%=roleXmlName%>">
	<xsd:complexType>
	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="1">
	<%
	for all types {
	%>
	<xsd:element name="<%=typeXmlName%>" type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>" />
	<%
	} // end for all types
	%>
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>

Description	A set of XML elements is created that represent the property: • A role wrapper XML element with maxOccurs=1. The name of the XML element is defined by the xml.namePlural of the property. • Role XML elements are nested within the role wrapper element. The minimum occurrence of these elements is the lower multiplicity of the property; the maximum occurrence is the upper multiplicity. • Nested in role element at most one XML-element representing the type is allowed.
UML example	See above

XML schema example	<xsd:element name="THE-BS" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:element name="THE-B"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:element name="B1" type="AR:B1"/> <xsd:element name="B2" type="AR:B2"/> </xsd:choice> </xsd:complexType> </xsd:element> </xsd:choice> </xsd:complexType> </xsd:element> <xsd:element name="THE-CS" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:element name="THE-C"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:element name="C" type="AR:C"/> </xsd:choice> </xsd:complexType> </xsd:element> </xsd:choice> </xsd:complexType> </xsd:element>
XML instance example	<...> <THE-BS> <THE-B> <B1> <ATT-B1>someValueB1</ATT-B1> </B1> <B2> <ATT-B2>someValueB2</ATT-B2> </B2> </THE-B> </THE-BS> <THE-CS> <THE-C> <C> <ATT-C>someValueC</ATT-C> </C> </THE-C> </THE-CS> </...>

5.4.3 [APRXML0023] Create composite property representation (1100) [Tc0010Sc1100]

Applies to	Property
Precondition	xml.roleWrapperElement == true && xml.roleElement == true && xml.typeWrapperElement == false && xml.typeElement == false

Target pattern	<% if (types.length > 1) { %>
	
	<xsd:element name="<%=roleXmlNamePlural%>"
	minOccurs="<% if (lowerMultiplicity>0) {%>1<% } else { %>0<%}%>" maxOccurs="1">
	<xsd:complexType>
	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>">
	<xsd:element name="<%=roleXmlName%>">
	<xsd:complexType>
	<xsd:choice minOccurs="<% if (lowerMultiplicity>0) {%>1<% } else { %>0<%}%>" maxOccurs="1">
	
	<%
	for all types {
	%>
	
	<xsd:group ref="<%=typeXmlNsPrefix%>:<%=typeXmlName%>"/>
	
	<% } // end for all types %>
	
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>
	
	<% } else { %>
	
	<xsd:element name="<%=roleXmlNamePlural%>"
	minOccurs="<% if (lowerMultiplicity>0) {%>1<% } else { %>0<%}%>" maxOccurs="1">
	<xsd:complexType>
	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>">
	<xsd:element name="<%=roleXmlName%>" type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>"/>
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>
	
	<% } %>

Description	A set of XML elements is created that represent the property: • A role wrapper XML element with maxOccurs=1. The name of the XML element is defined by the xml.namePlural of the property. • Role XML elements are nested within the role wrapper element. The minimum occurrence of these elements is the lower multiplicity of the property; the maximum occurrence is the upper multiplicity. The content model of the type directly shows up within the declaration of this element.
UML example	See above

XML schema example	<xsd:element name="THE-BS" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:element name="THE-B"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:group ref="AR:B1"/> <xsd:group ref="AR:B2"/> </xsd:choice> </xsd:complexType> </xsd:element> </xsd:choice> </xsd:complexType> </xsd:element> <xsd:element name="THE-CS" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:element name="THE-C" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:group ref="AR:C"/> </xsd:choice> </xsd:complexType> </xsd:element> </xsd:choice> </xsd:complexType> </xsd:element>
XML instance example	<…> <THE-BS> <THE-B> <ATT-B1>someValueB1</ATT-B1> </THE-B> <THE-B> <ATT-B2>someValueB2</ATT-B2> </THE-B> </THE-BS> <THE-CS> <THE-C> <ATT-C>someValueC</ATT-C> </THE-C> </CS> </THE-CS> </…>

5.4.4 [APRXML0022] Create composite property representation (1011) [Tc0010Sc1011]

Applies to	Property
Precondition	xml.roleWrapperElement == true && xml.roleElement == false && xml.typeWrapperElement == true && xml.typeElement == true

Target pattern	<xsd:element name="<%=roleXmlNamePlural%>"
	minOccurs="<%if (lowerMultiplicity>0) {%>1<%} else {%>0<%}%>"
	maxOccurs="1">
	<xsd:complexType>
	<xsd:all minOccurs="<%if (lowerMultiplicity>0) {%>1<%} else {%>0<%}%>"
	maxOccurs="<%=types.length%>">
	<%
	for all types {
	%>
	<xsd:element name="<%=typeXmlNamePlural%>" minOccurs="0" maxOccurs="1">
	<xsd:complexType>
	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>">
	<xsd:element name="<%=typeXmlName%>"
	type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>"/>
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>
	<% } // end for all types %>
	</xsd:all>
	</xsd:complexType>
	</xsd:element>

Description	A set of XML elements is created that represent the property: • A role wrapper XML element with maxOccurs=1. The name of the XML element is defined by the xml.namePlural of the property. • For each type and subtype of the property a type wrapper XML element is created. • Nested in these type wrappers only elements representing the same types as the type wrapper are allowed.
UML example	See above

XML schema example	<xsd:element name="THE-BS" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:all minOccurs="0" maxOccurs="2"> <xsd:element name="B1S" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:element name="B2" type="AR:B2"/> </xsd:choice> </xsd:complexType> </xsd:element> <xsd:element name="B2S" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:element name="B2" type="AR:B2"/> </xsd:choice> </xsd:complexType> </xsd:element> </xsd:all> </xsd:complexType> </xsd:element> <xsd:element name="THE-CS" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:all minOccurs="0" maxOccurs="1"> <xsd:element name="CS" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:element name="C" type="AR:C"/> </xsd:choice> </xsd:complexType> </xsd:element> </xsd:all> </xsd:complexType> </xsd:element>
XML instance example	<...> <THE-BS> <B1S> <B1> <ATT-B1>someValueB1</ATT-B1> </B1> </B1S> <B2S> <B2> <ATT-B2>someValueB2</ATT-B2> </B2> </B2S> </THE-BS> <THE-CS> <CS> <C> <ATT-C>someValueC</ATT-C> </C> </CS> </THE-CS> </...>

5.4.5 [APRXML0010] Create composite property representation (1001) [Tc0010Sc1001]

Applies to	Property
Precondition	xml.roleWrapperElement == true && xml.roleElement == false && xml.typeWrapperElement == false && xml.typeElement == true

Target pattern	<%=roleXmlNamePlural%>
	minOccurs="<%if (lowerMultiplicity>0) {%>1<%} else {%>0<%}%>"
	maxOccurs="1">
	<xsd:complexType>
	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>">
	<%
	for all types {
	%>
	<xsd:element name="<%=typeXmlName%>" type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>"/>
	<%
	} // end for all types
	%>
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>

Description	A set of XML elements is created that represent the property: • A role wrapper XML element with maxOccurs=1. The name of the XML element is defined by the xml.namePlural of the property. • An XML-element representing the type or subtype of the property. The name of this element is defined by the xml.name of the type of the property.
UML example	See above
XML schema example	<xsd:element name="THE-BS" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:element name="B1" type="AR:B1"/> <xsd:element name="B2" type="AR:B2"/> </xsd:choice> </xsd:complexType> </xsd:element> <xsd:element name="THE-CS" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:element name="C" type="AR:C"/> </xsd:choice> </xsd:complexType> </xsd:element>
XML instance example	<…> <THE-BS> <B1> <ATT-B1>someValueB1</ATT-B1> </B1> <B2> <ATT-B2>someValueB2</ATT-B2> </B2> </THE-BS> <THE-CS> <C> <ATT-C>someValueC</ATT-C> </C> </THE-CS> </…>

5.4.6 [APRXML0011] Create composite property representation (0111) [Tc0010Sc0111]

Applies to	Property
Precondition	xml.roleWrapperElement == false && xml.roleElement == true && xml.typeWrapperElement == true && xml.typeElement == true

Target pattern	<xsd:element name="<%=roleXmlName%>" minOccurs="<%=lowerMultiplicity%>"
	maxOccurs="<%=upperMultiplicity%>">
	<xsd:complexType>
	<xsd:all minOccurs="<%if (lowerMultiplicity.equals("0")) {%>0<%} else {%>1<%}%>">
	<%
	for all types {
	%>
	<xsd:element name="<%=typeXmlNamePlural%>" minOccurs="0" maxOccurs="1">
	<xsd:complexType>
	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>">
	<xsd:element name="<%=typeXmlName%>"
	type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>"/>
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>
	<%
	} // end for all types
	%>
	</xsd:all>
	</xsd:complexType>
	</xsd:element>

Description	A set of XML elements is created that represent the property: • XML elements representing the property name. The name of these XML-elements is defined by the xml.name of the property. The minimum occurrence of these elements is the lower multiplicity of the property; the maximum occurrence is the upper multiplicity. • For each type and subtype of the property a type wrapper XML element is created. The name of the XML element is defined by the xml.namePlural of the type of the property. • Nested in these type wrappers only elements representing the same types as the type wrapper are allowed.
UML example	See above

XML schema example	<xsd:element name="THE-B" minOccurs="0" maxOccurs="unbounded"> <xsd:complexType> <xsd:all minOccurs="0"> <xsd:element name="B1S" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:element name="B1" type="AR:B1"/> </xsd:choice> </xsd:complexType> </xsd:element> <xsd:element name="B2S" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:element name="B2" type="AR:B2"/> </xsd:choice> </xsd:complexType> </xsd:element> </xsd:all> </xsd:complexType> </xsd:element> <xsd:element name="THE-C" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:all minOccurs="0"> <xsd:element name="CS" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:element name="C" type="AR:C"/> </xsd:choice> </xsd:complexType> </xsd:element> </xsd:all> </xsd:complexType> </xsd:element>
XML instance example	<...> <THE-B> <B1S> <B1> <ATT-B1>someValueB1</ATT-B1> </B1> </B1S> <B2S> <B2> <ATT-B2>someValueB2</ATT-B2> </B2> </B2S> </THE-B> <THE-C> <CS> <C> <ATT-C>someValueC</ATT-C> </C> </CS> </THE-C> </...>

5.4.7 [APRXML0012] Create composite property representation (0101) [Tc0010Sc0101]

Applies to	Property
Precondition	xml.roleWrapperElement == false && xml.roleElement == true && xml.typeWrapperElement == false && xml.typeElement == true

Target pattern	<%=roleXmlName%>" minOccurs="<%=lowerMultiplicity%>"
	maxOccurs="<%=upperMultiplicity%>">
	<xsd:complexType>
	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="1">
	<%
	for all types {
	%>
	<xsd:element name="<%=typeXmlName%>" type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>"/>
	<%
	} // end for all types
	%>
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>

Description	A set of XML elements is created that represent the property: • XML elements representing the property name. The name of these XML-elements is defined by the xml.name of the property. The minimum occurrence of these elements is the lower multiplicity of the property; the maximum occurrence is the upper multiplicity. • Nested in role XML element at most one XML-element representing the type is allowed.
UML example	See above
XML schema example	<xsd:element name="THE-B" minOccurs="0" maxOccurs="unbounded"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:element name="B1" type="AR:B1"/> <xsd:element name="B2" type="AR:B2"/> </xsd:choice> </xsd:complexType> </xsd:element> <xsd:element name="THE-C" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:element name="C" type="AR:C"/> </xsd:choice> </xsd:complexType> </xsd:element>
XML instance example	<…> <THE-B> <B1> <ATT-B1>someValueB1</ATT-B1> </B1> </THE-B> <THE-B> <B2> <ATT-B2>someValueB2</ATT-B2> </B2> </THE-B> <THE-C> <C> <ATT-C>someValueC</ATT-C> </C> </THE-C> </…>

5.4.8 [APRXML0013] Create composite property representation (0100) [Tc0010Sc0100]

Applies to	Property
Precondition	xml.roleWrapperElement == false && xml.roleElement == true && xml.typeWrapperElement == false && xml.typeElement == false

Target pattern	<%
	if (types.length > 1) {
	%>
	
	<xsd:element name="<%=roleXmlName%>" minOccurs="<%=lowerMultiplicity%>"
	maxOccurs="<%=upperMultiplicity%>">
	<xsd:complexType>
	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="1">
	
	<%
	for all types {
	%>
	
	<xsd:group ref="<%=typeXmlNsPrefix%>:<%=typeXmlName%>"/>
	
	<%
	} // end for all types
	%>
	
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>
	
	<%
	} else {
	
	// NOT type.length >1
	%>
	
	<xsd:element name="<%=roleXmlName%>" type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>"
	minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>"/>
	
	<%
	}
	%>

Description	An XML element is created that represents the property: The name of the XML element is defined by the xml.name of the property. The minimum occurrence of this element is the lower multiplicity of the property; the maximum occurrence is the upper multiplicity. The content model of the type directly shows up within the declaration of this element.
UML example	See above
XML schema example	<xsd:element name="THE-B" minOccurs="0" maxOccurs="unbounded"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:group ref="AR:B1"/> <xsd:group ref="AR:B2"/> </xsd:choice> </xsd:complexType> </xsd:element> <xsd:element name="THE-C" type="AR:C" minOccurs="0" maxOccurs="1"/>

XML instance example	<…> <THE-B> <ATT-B1>someValueB1</ATT-B1> </THE-B> <THE-B> <ATT-B2>someValueB2</ATT-B2> </THE-B> <THE-C> <ATT-C>someValueC</ATT-C> </THE-C> </…>

5.4.9 [APRXML0014] Create composite property representation (0011) [Tc0010Sc0011]

Applies to	Property
Precondition	xml.roleWrapperElement == false && xml.roleElement == false && xml.typeWrapperElement == true && xml.typeElement == true

Target pattern	<%
	for all types {
	%>
	
	<xsd:element name="<%=typeXmlNamePlural%>"
	minOccurs="<%if (lowerMultiplicity>0) {%>1<%} else {%>0<%}%>"
	maxOccurs="1">
	<xsd:complexType>
	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>">
	<xsd:element name="<%=typeXmlName%>" type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>"/>
	</xsd:choice>
	</xsd:complexType>
	</xsd:element>
	
	<%
	} // end for all types
	%>

Description	A set of XML elements is created that represent the property: • For each type and subtype of the property a type wrapper XML element is created. The name of the XML element is defined by the xml.namePlural of the type of the property. • Nested in these type wrappers only elements representing the same types as the type wrapper are allowed.
UML example	See above

XML schema example	<xsd:element name="B1S" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:element name="B1" type="AR:B1"/> </xsd:choice> </xsd:complexType> </xsd:element> <xsd:element name="B2S" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:element name="B2" type="AR:B2"/> </xsd:choice> </xsd:complexType> </xsd:element> <xsd:element name="CS" minOccurs="0" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:element name="C" type="AR:C"/> </xsd:choice> </xsd:complexType> </xsd:element>
XML instance example	<...> <B1S> <B1> <ATT-B1>someValueB1</ATT-B1> </B1> </B1S> <B2S> <B2> <ATT-B2>someValueB2</ATT-B2> </B2> </B2S> <CS> <C> <ATT-C>someValueC</ATT-C> </C> </CS> </...>

5.4.10 [APRXML0015] Create composite property representation (0001) [Tc0010Sc0001]

Applies to	Property
Precondition	xml.roleWrapperElement == false && xml.roleElement == false && xml.typeWrapperElement == false && xml.typeElement == true

Target pattern	<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>">
	
	<%
	for all types {
	%>
	
	<xsd:element name="<%=typeXmlName%>" type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>"/>
	
	<%
	} // end for all types
	%>
	
	</xsd:choice>

Description	An XML element is created that represents the property: The name of the XML element is defined by the xml.name of the type of the property. The minimum occurrence of this element is the lower multiplicity of the property; the maximum occurrence is the upper multiplicity. The content model of the type directly shows up within the declaration of this element.
UML example	See above

XML schema example	`<xsd:choice minOccurs="0" maxOccurs="unbounded">`
	`<xsd:element name="B1" type="AR:B1"/>`
	`<xsd:element name="B2" type="AR:B2"/>`
	`</xsd:choice>`
	
	`<xsd:choice minOccurs="0" maxOccurs="1">`
	`<xsd:element name="C" type="AR:C"/>`
	`</xsd:choice>`

XML instance example	`<...>`
	
	`<B1>`
	`<ATT-B1>someValueB1</ATT-B1>`
	`</B1>`
	
	`<B2>`
	`<ATT-B2>someValueB2</ATT-B2>`
	`</B2>`
	
	`<C>`
	`<ATT-C>someValueC</ATT-C>`
	`</C>`
	
	`</...>`

5.4.11 [APRXML0016] Create composite property representation (0000) [Tc0010Sc0000]

Applies to	Property
Precondition	xml.roleWrapperElement == false && xml.roleElement == false && xml.typeWrapperElement == false && xml.typeElement == false

Target pattern	`<xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>">`
	
	`<%`
	`for all types {`
	`%>`
	
	`<xsd:group ref="<%=typeXmlNsPrefix%>:<%=typeXmlName%>"/>`
	
	`<%`
	`} // end for all types`
	`%>`
	
	`</xsd:choice>`

Description	No XML element is defined for the property. The content model of the type and all subtypes of the property is inserted directly in the xsd:group that represents the class that owns the property.
UML example	See above

XML schema example	<xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:group ref="AR:B1"/> <xsd:group ref="AR:B2"/> </xsd:choice> <xsd:choice minOccurs="0" maxOccurs="1"> <xsd:group ref="AR:C"/> </xsd:choice>
XML instance example	<…> <ATT-B1>someValueB1</ATT-B1> <ATT-B2>someValueB2</ATT-B2> <ATT-C>someValueC</ATT-C> </…>

5.5 Create reference representation [Tc0012]

Figure 5-6 shows an overview on mapping rules for references (properties with aggregation=none). References are always mapped to XML-elements. If the tagged value xml.propertyWrapperElement=true is applied to the property, then a wrapper element is created for the reference.

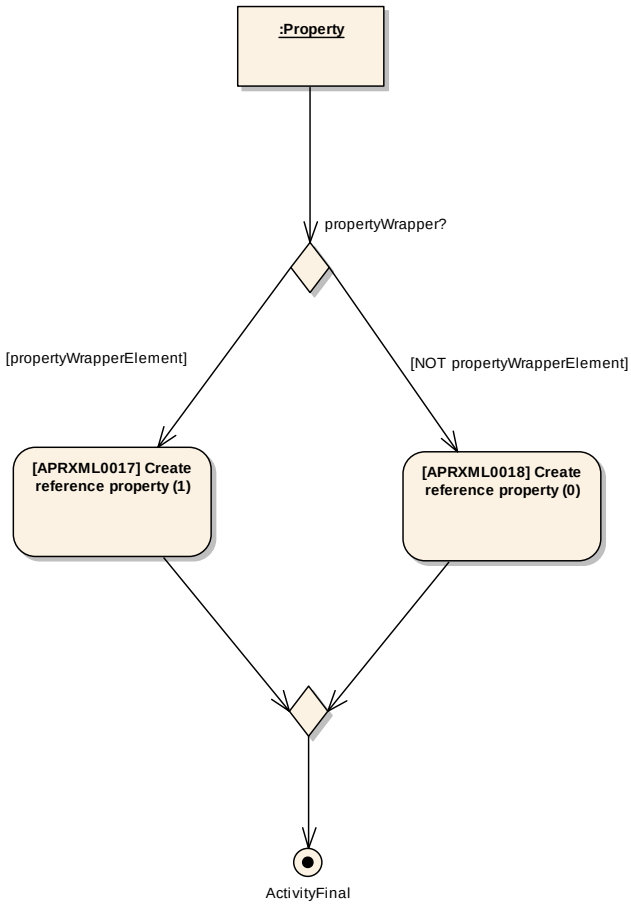


Figure 5-6: Property representation (aggregation = none)

Figure 5-7 shows an example metamodel that uses a reference. This example is used to illustrate the following two mapping rules.

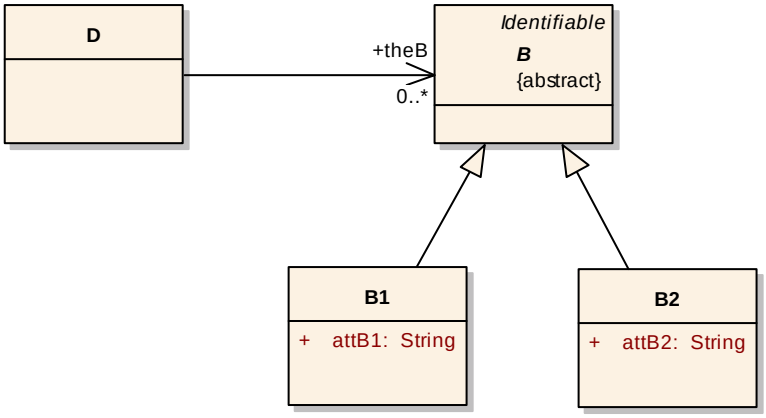


Figure 5-7: Example metamodel using a reference

Figure 5-8 shows an example instance of a reference. This example is used to illustrate the following two mapping rules.

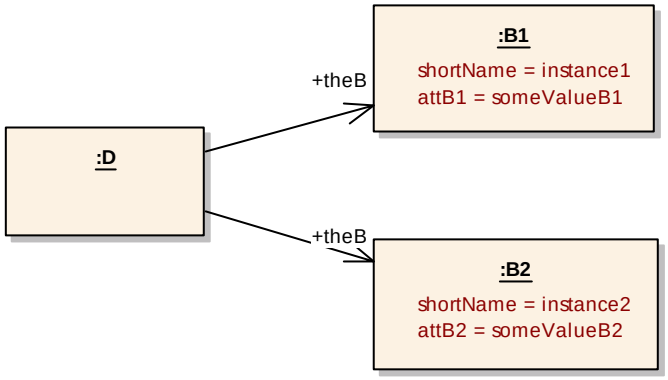


Figure 5-8: Example instance of reference

5.5.1 [APRXML0017] Create reference property representation (1) [Tc0012Sc1]

Applies to	Property
Precondition	xml.roleWrapperElement=true

Target pattern	<xsd:element name="<%=roleXmlNamePlural%>" minOccurs="<%=lowerMultiplicity%>" maxOccurs="1"> <xsd:complexType> <xsd:choice minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>"> <xsd:element name="<%=roleXmlName%>"> </xsd:complexType> <xsd:simpleContent> <xsd:extension base="AR:REF"> <xsd:attribute name="DEST" type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>--SUBTYPE-ENUM" use="required"/> </xsd:extension> </xsd:simpleContent> </xsd:complexType> </xsd:element> </xsd:choice> </xsd:complexType> </xsd:element>
Description	The following XML-elements represent a property (if aggregation=none): • A XML element that wraps several XML-element of the same XML-name. The name of the wrapper is defined by the xml.namePlural of the property. Please not the default xml.namePlural is defined by the default singular XML name appended by "-REFS", "-TREFS" (see section 4.2.3). • An XML element that represents the reference itself. The name of this element is defined by the xml.name of the property. Please not the default xml.name is defined by the default singular XML name appended by "-REF", "-TREF" (see section 4.2.3).
UML example	See above
XML schema example	<xsd:element name="THE-B-REFS" > <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:element name="THE-B-REF" type="AR:REF" minOccurs="0" maxOccurs="unbounded"> <xsd:complexType> <xsd:simpleContent> <xsd:extension base="AR:REF"> <xsd:attribute name="DEST" type="B--SUBTYPES-ENUM" use="required"/> </xsd:extension> </xsd:simpleContent> </xsd:complexType> </xsd:element> </xsd:choice> </xsd:complexType> </xsd:element>
XML instance example	<…> <THE-B-REFS> <THE-B-REF DEST="B-1">instance1</THE-B-REF> <THE-B-REF DEST="B-2">instance2</THE-B-REF> </THE-B-REFS> </…>

5.5.2 [APRXML0018] Create reference property representation (0) [Tc0012Sc2]

Applies to	Property
Precondition	Xml.roleWrapperElement=false

Target pattern	<xsd:element name="<%=roleXmlName%>" minOccurs="<%=lowerMultiplicity%>" maxOccurs="<%=upperMultiplicity%>"> <xsd:complexType> <xsd:simpleContent> <xsd:extension base="AR:REF"> <xsd:attribute name="DEST" type="<%=typeXmlNsPrefix%>:<%=typeXmlName%>--SUBTYPES-ENUM" use="required"/> </xsd:extension> </xsd:simpleContent> </xsd:complexType> </xsd:element>
Description	An XML element represents the reference. The name of this element is defined by the xml.name of the property. Please not the default xml.name is defined by the default singular XML name appended by "-REF", "-TREF" (see section 4.2.3).
UML example	See above
XML schema example	<xsd:element name="THE-B-REF" type="AR:REF" minOccurs="0" maxOccurs="unbounded"> <xsd:complexType> <xsd:simpleContent> <xsd:extension base="AR:REF"> <xsd:attribute name="DEST" type="B--SUBTYPES-ENUM" use="required"/> </xsd:extension> </xsd:simpleContent> </xsd:complexType> </xsd:element>
XML instance example	<…> <THE-B-REF DEST="B-1">instance1</THE-B-REF> <THE-B-REF DEST="B-2">instance2</THE-B-REF> </…>

5.6 Create predefined data types [Tc0007]

5.6.1 [APRXML0020] Create AR:IDENTIFIER

An Identifier is a special form of a String, which basically has to satisfy the requirements for names used in typical programming languages. Only the following characters are allowed in identifiers: a-z, A-Z, 0-9, _.
Additionally the minimum length of an identifier is 1, the maximum length is 128 and it has to start with a letter.

```
<xsd:simpleType name="IDENTIFIER">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[a-zA-Z][a-zA-Z0-9_]{0,127}"/>
  </xsd:restriction>
</xsd:simpleType>
```

5.6.2 [APRXML0021] Create AR:REF

References are sequences of IDENTIFIERs separated by "/".

```
<xsd:simpleType name="REF">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="(/[a-zA-Z][a-zA-Z0-9_]{0,127})+"/>
  </xsd:restriction>
</xsd:simpleType>
```

6 XML description production

6.1 Introduction

This chapter specifies rules for the serialization of AUTOSAR models (instances of the AUTOSAR metamodel) to AUTOSAR XML descriptions. The inverse of these rules can be applied to an AUTOSAR XML description to reconstruct the AUTOSAR model.
The production rules are provided as a specification of the AUTOSAR XML description production and consumption process. They should not be viewed as precibing any particular algorithm for XML producer or consumer implementations.

6.2 Overall document structure

The AUTOSAR XML schema is monolitic and defines one single XML namespace (http://autosar.org/<version>) for all its XML elements and XML attributes. (<version> corresponds to the version of the AUTOSAR release). Additionally the AUTOSAR XML schema does not contain any anchor for integrating XML elements or attributes defined in other XML schemas or XML namespaces[4].
In order to simplify the implementation of the first AUTOSAR tools, the AUTOSAR XML namespace (http://autosar.org/<version>) SHALL be used as the default XML namespace[5]. Additionally the AUTOSAR XML descriptions SHALL be UTF-8 encoded.

AUTOSAR XML descriptions SHALL use the following XML document header. The namespace contains information of the AUTOSAR release. Encoding the release information into the namespace allows for parallel use of different versions of XML schema and XML descriptions.
E.g. the XML namespace http://autosar.org/2.1.0 corresponds to the AUTOSAR XML schema of the AUTOSAR release 2.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/2.1.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://autosar.org/2.1.0 autosar.xsd">

<!—the topLevelPackages go here -->

</AUTOSAR>
```

AUTOSAR XML descriptions may be shipped with or without the AUTOSAR XML schema. Therefore all AUTOSAR tools that validate AUTOSAR XML descriptions against the AUTOSAR XML schema SHALL have their own copy of the supported AUTOSAR XML schema. They SHALL be able to find their local copy of the XML

[4] The only exception is the namespace http://www.w3.org/2001/XMLSchema-instance and the namespace prefix xsi.
[5] It is higly recommended to support the XML namespace concepts for tools that are intended to interpret or produce AUTOSAR XML descriptions. Later versions of the AUTOSAR XML schema might be more modular and might use more than one XML namespace.

schema file without explicitly setting the xsi:schemaLocation field in the AUTOSAR XML description.
In other words:

- The field xsi:schemaLocation is optional
- AUTOSAR tools SHALL continue interpreting AUTOSAR XML descriptions even if the path defined in the xsi:schemaLocation is not valid.

6.3 Naming convention of AUTOSAR XML description files

AUTOSAR XML descriptions SHALL use the file extension ".arxml" (short for AUTOSAR XML).
Additionally it is recommended to use the following naming convention for the AUTOSAR XML descriptions:

<identifier>_<use-case>.arxml

The identifier describes the content of the AUTOSAR XML description. The use-case defines the intended usage of the contained information. AUTOSAR has not defined formally defined the usecases for AUTOSAR XML description filenames. However, some examples are:

- swc: Software component
- sys: System description
- ecuc: ECU configuration

The maximum length of the filename is restricted to 255 characters. The following restrictions apply:

- Identifier ::= [a-zA-Z][a-zA-Z0-9_]{0,240}
- Use-case ::= [a-zA-Z][a-zA-Z0-9]{0,8}

The filenames are not case-sensitive.

6.4 Linking

Due to the complexity of relative SHORT-NAME references only absolute SHORT-NAME references are allowed.

7 Compliance

7.1 AUTOSAR XML schema compliance

AUTOSAR XML Schemas must be equivalent to those generated by the AUTOSAR XML Schema production rules specified in this document. Equivalence means that:

- XML documents that are valid under the AUTOSAR XML Schema would be valid in a conforming XML Schema
- and that those XML documents that are not valid under the AUTOSAR XML Schema are not valid in a conforming XML Schema.

7.2 AUTOSAR XML document compliance

The XML document must be “valid” and “well-formed” as defined by the XML recommendation, whether used with or without the document’s corresponding AUTOSAR XML Schema(s). Although it is optional not to transmit and/or validate a document with its AUTOSAR XML Schema(s), the document must still conform as if the check had been made.

8 References

8.1 Normative references to AUTOSAR documents

[1] Glossary
AUTOSAR_Glossary.pdf.

[2] Specification of Interoperability of Authoring Tools
AUTOSAR_InteroperabilityOfAuthoringTools.pdf

[3] Metamodel,
AUTOSAR_MetaModel.EAP

[4] Template UML Profile and Modeling Guide
AUTOSAR_TemplateModelingGuide.pdf

8.2 Normative references to external documents

[5] XML Metadata Interchange (XMI) Specification version 2.1,
http://www.omg.org/cgi-bin/apps/doc?formal/05-09-01.pdf

[6] XML Metadata Interchange (XMI) Specification version 1.2,
http://www.omg.org/cgi-bin/apps/doc?formal/02-01-01.pdf

[7] XML Information Set (Second Edition), W3C Recommendation 4 February 2004,
http://www.w3.org/TR/xml-infoset/

[8] XML Schema 1.1,
http://www.w3.org/XML/Schema

[9] Extensible Markup Language (XML) 1.1,
http://www.w3.org/TR/xml11/

[10] Namespaces in XML 1.1, W3C Recommendation 4 February 2004,
http://www.w3.org/TR/xml-names11/

[11] MSR-TR-CAP,
http://www.msr-wg.de/medoc/download/msr-tr-cap/msr-tr-cap.pdf

[12] MSR-SW,
http://www.msr-wg.de/medoc/download/msrsw/v230/msrsw_v230-eadoc-en/msrsw_v2_3_0.sl-eadoc.pdf

[13] XHTML,
http://www.w3.org/TR/xhtml11/

[14] Unified Modeling Language: Superstructure, Version 2.0, OMG Available Specification, ptc/05-07-04.
http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf

[15] Unified Modeling Language: Infrastructure, Version 2.0. OMG Adopted Specification, ptc/03-09-15.
http://www.omg.org/cgi-bin/apps/doc?ptc/03-09-15.pdf

[16] Unified Modeling Language OCL, Version 2.0, OMG Available Specification, ptc/05-06-06.
http://www.omg.org/cgi-bin/apps/doc?ptc/05-06-06.pdf

[17] Meta-Object Facility MOF, Version 2.0, OMG Available Specification, ptc/04-10-15.
http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-15.pdf

8.3 Other references

[18] Carlson, David; Modeling XML Applications with UML, Practical e-Business Applications; Addison Wesley; 2001.