Document Title	AUTOSAR XML Schema Production Rules
Document Category	Template specification (TPS)
Document UID	122
Document Status	published
Part of AUTOSAR Standard	Foundation
Part of Standard Release	R26-11

Document Change History

Date	Release	Changed by	Description
2026-11-26	R26-11	AUTOSAR Release Management	• No content changes
2025-11-27	R25-11	AUTOSAR Release Management	• Distinct production of InstanceRefs: different handling required for abstract InstanceRefs and for InstanceRefs with abstract target class
2024-11-27	R24-11	AUTOSAR Release Management	• Editorial changes
2023-11-23	R23-11	AUTOSAR Release Management	• Editorial changes
2022-11-24	R22-11	AUTOSAR Release Management	• clarified usage of the term “properties” • clarified when to produce xsd:groups • minor corrections / clarifications / editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• minor corrections / clarifications / editorial changes
2020-11-30	R20-11	AUTOSAR Release Management	• allow additional property configuration, see section 3.1.2.3 • minor corrections / clarifications / editorial changes
2019-11-28	R19-11	AUTOSAR Release Management	• Editorial changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes

▽

△

2017-12-08	4.3.1	AUTOSAR Release Management	• minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2016-11-30	4.3.0	AUTOSAR Release Management	• Renamed Document • Removed chapter "6 XML description production rules" • Removed section about XML description conformance from chapter 7
2015-07-31	4.2.2	AUTOSAR Release Management	• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2014-10-31	4.2.1	AUTOSAR Release Management	• Formal adaptations concerning traceability
2014-03-31	4.1.3	AUTOSAR Release Management	• Minor corrections concerning XML namespace
2013-10-31	4.1.2	AUTOSAR Release Management	• Added tabular overview of default configuration of schema generator ([TPS_XMLSPR_00056])
2013-03-15	4.1.1	AUTOSAR Administration	• Removed references to "Template UML Profile and Modeling Guide" • Changed chapter 3.2.4.1
2011-12-22	4.0.3	AUTOSAR Administration	• Formal adaptations concerning traceability • Harmonized naming proposal for arxml files with AUTOSAR_TR_ InteroperabilityOfAutosarTools • Updated XML Persistence mechanism regarding primitive types with attributes
2010-09-30	3.1.5	AUTOSAR Administration	• Added description of tag default configuration for association without stereotype (chapter 3.2.3.1) • Enhanced description of tag 'xml.xsd.customType'

▽

△

2010-02-02	3.1.4	AUTOSAR Administration	• Modeling and handling of primitive types has been revised • Inheritance information is visible in schema now for all superclasses, also for empty abstract classes • Variant Handling is handled in Generic Structure Template • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Document meta information extended • Small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• "Advice for users" revised • "Revision Information" added
2006-11-28	2.1	AUTOSAR Administration	• Updated instanceRef references • Only absolute paths allowed • Naming of instanceRef • Destination type of references • Version info in namespace • Legal disclaimer revised
2006-05-16	2.0	AUTOSAR Administration	• Initial Release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction 9

2 XML Schema design principles 11
2.1 Notes on UML2.0 semantics of the AUTOSAR meta-model 11
2.1.1 Representation of association (aggregation = composite) 11
2.1.2 Representation of attribute (aggregation = composite) 12
2.1.3 Representation of associations (aggregation = none) 13
2.1.4 Representation of attributes (aggregation = none) 14
2.2 Notes on use of W3C XML schema 15
2.3 Handling Inheritance 16
2.4 Generic Approach 16
2.5 XML element versus attribute 17
2.6 XML names 17
2.7 Order of XML-elements 18
2.7.1 Order of xml elements 18
2.7.2 Order of xml elements of derived uml:properties 19
2.8 Linking 22
2.9 Transmitting incomplete Data 22
2.10 Identification of XML schema version in XML descriptions 22

3 Configuration of XML schema production 23
3.1 Tailoring schema production 23
3.1.1 Overview 23
3.1.2 Constraints on tags 28
3.1.2.1 Constraints on tags applied to properties 29
3.1.2.2 Constraints on tags applied to classes 30
3.1.2.3 Constraints on values of xml.*Element tagged values 30
3.2 Default configuration of XML schema production 32
3.2.1 Configuration of multiplicities 33
3.2.2 Mapping configuration for properties 33
3.2.3 Mapping configuration for references 37
3.2.3.1 References without stereotypes 37
3.2.3.2 Instance references 38
3.2.3.3 References with stereotype ≪isOfType≫ 41
3.2.4 Stereotypes applied to classes 42
3.2.4.1 Stereotype ≪atpMixed≫ 42
3.2.4.2 Stereotype ≪atpMixedString≫ 42

4 XML Schema production rules 44
4.1 Create model representation 44
4.1.1 Create xsd:schema 45
4.2 Create class representation 46
4.2.1 Create xsd:group 47

4.2.2 Create xsd:attributeGroup . . . 49
4.2.3 Create xsd:complexType . . . 50
4.2.4 Create xsd:complexType with simple content . . . 51
4.2.5 Create global xsd:element . . . 52
4.2.6 Create enumeration of subtypes . . . 53
4.2.7 Create reference to XML predefined data type . . . 53
4.2.8 Create a custom simple type . . . 54
4.2.9 Create xsd:simpleType for enumeration . . . 55
4.3 Create composite property representation (mapping to XML attributes) . 56
4.3.1 Create xsd:attribute . . . 56
4.4 Create composite property representation (mapping to XML elements) . 57
4.4.1 Create composite property representation (1111) . . . 59
4.4.2 Create composite property representation (1101) . . . 62
4.4.3 Create composite property representation (1100) . . . 63
4.4.4 Create composite property representation (1011) . . . 65
4.4.5 Create composite property representation (1000) . . . 67
4.4.6 Create composite property representation (1001) . . . 68
4.4.7 Create composite property representation (0111) . . . 69
4.4.8 Create composite property representation (0101) . . . 71
4.4.9 Create composite property representation (0100) . . . 72
4.4.10 Create composite property representation (0011) . . . 73
4.4.11 Create composite property representation (0001) . . . 74
4.4.12 Create composite property representation (0000) . . . 75
4.5 Create reference representation . . . 76
4.5.1 Create reference property representation (1) . . . 78
4.5.2 Create reference property representation (0) . . . 79
4.5.3 Create a reference to attributes in foreign namespaces . . . 80
5 AUTOSAR XML schema compliance 82
A Change history of AUTOSAR traceable items 83
A.1 Specification Item Mapping: Document migration from TR to TPS . . . 83
A.2 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 84
A.2.1 Added Specification Items in R26-11 . . . 84
A.2.2 Changed Specification Items in R26-11 . . . 84
A.2.3 Deleted Specification Items in R26-11 . . . 84
A.3 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 85
A.3.1 Added Specification Items in R25-11 . . . 85
A.3.2 Changed Specification Items in R25-11 . . . 85
A.3.3 Deleted Specification Items in R25-11 . . . 85
A.4 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 85
A.4.1 Added Specification Items in R24-11 . . . 85

A.4.2 Changed Specification Items in R24-11 85
A.4.3 Deleted Specification Items in R24-11 85
A.5 Traceable item history of this document according to AUTOSAR Release R23-11 86
A.5.1 Added Specification Items in R23-11 86
A.5.2 Changed Specification Items in R23-11 86
A.5.3 Deleted Specification Items in R23-11 86
A.6 Traceable item history of this document according to AUTOSAR Release R22-11 86
A.6.1 Added Specification Items in R22-11 86
A.6.2 Changed Specification Items in R22-11 86
A.6.3 Deleted Specification Items in R22-11 87
A.7 Traceable item history of this document according to AUTOSAR Release R21-11 87
A.7.1 Added Specification Items in R21-11 87
A.7.2 Changed Specification Items in R21-11 87
A.7.3 Deleted Specification Items in R21-11 87

References

[1] XML Schema 1.0
http://www.w3.org/TR/xmlschema-1

[2] Meta Model
AUTOSAR_FO_MMOD_MetaModel

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

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

[5] 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

[6] 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

[7] ARXML Serialization Rules
AUTOSAR_FO_TPS_ARXMLSerializationRules

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

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

[10] Generic Structure Template
AUTOSAR_FO_TPS_GenericStructureTemplate

[11] 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

1 Introduction

The AUTOSAR meta-model 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 [1] compliant XML schema can be compiled out of the AUTOSAR meta-model [2]. Using the production rules a new XML schema can be generated automatically whenever the meta-model is updated. The schema production rules defined in this document exceed the configuration possibilities of comparable approaches like XMI [3][4] 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 specification items.

Figure 1.1 describes the XML schema production rules in the overall context. The meta-levels 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-meta-level (M3). The **AUTOSAR template profile** [2] defines, which parts of UML2.0 are allowed to be used in the AUTOSAR meta-model.
- The **AUTOSAR meta-model** [2] is a UML2.0 [5] model that defines the language for describing AUTOSAR systems. The AUTOSAR meta-model is a graphical representation of a template. UML2.0 class diagrams are used to describe the attributes and their interrelationships. Stereotypes and OCL [6] (object constraint language) are used for defining specific semantics and constraints.
- An **AUTOSAR model** is an instance of the AUTOSAR meta-model. The information contained in the AUTOSAR model can be anything that is representable according to the AUTOSAR meta-model.

The meta-levels of the XML language are described on the right side of Figure 1.1:

- The **W3C XML schema specification** [1] 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 meta-model 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 meta-model is mapped to the AUTOSAR XML schema by means of the **XML schema production rules**.

The mapping between AUTOSAR User Models and AUTOSAR XML Descriptions is described in the **ARXML Serialization Rules** [7].

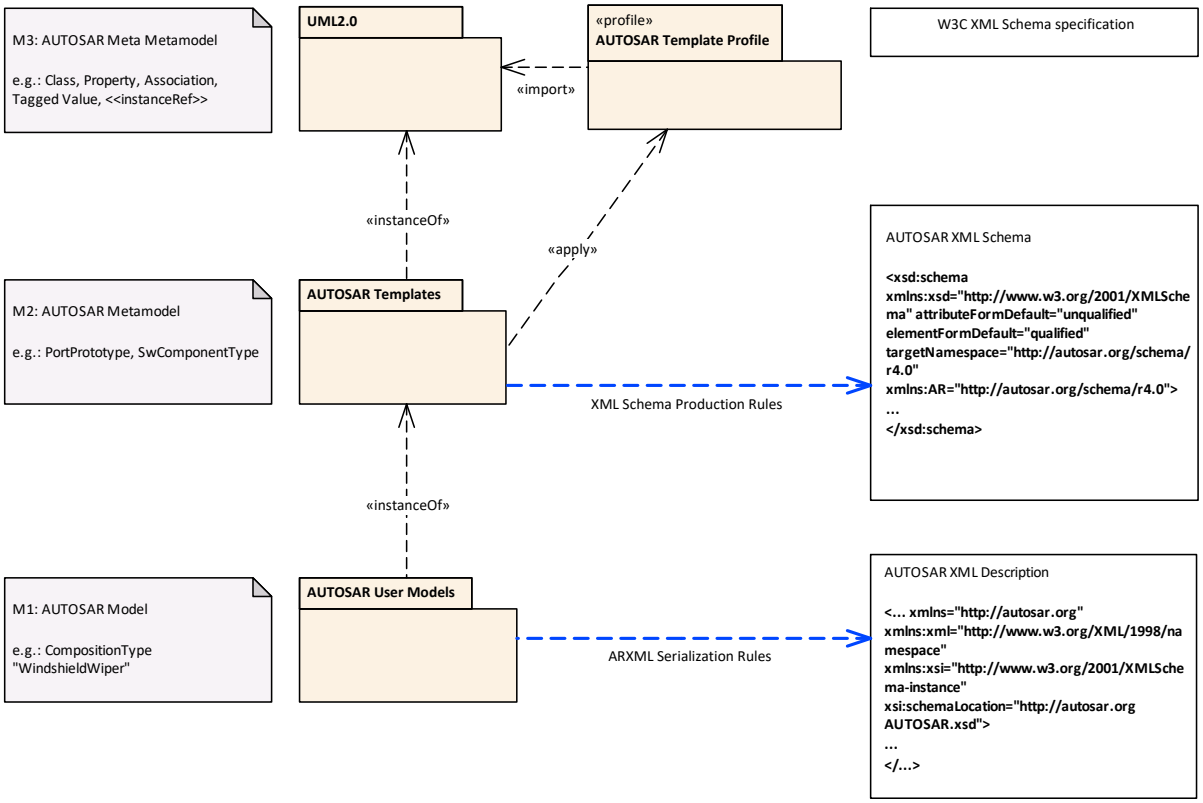

Figure 1.1: Context of XML schema production rules

This document is structured as follows:

- Chapter 1 (this chapter) describes the model XML schema production rules in the overall context of the AUTOSAR meta-model and the XML language.
- Chapter 2 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 3 describes how the XML schema production rules can be parameterized by means of tagged values. Additionally a default configuration for mapping the AUTOSAR meta-model to the AUTOSAR XML schema is given.
- Chapter 4 describes the XML schema production rules in more detail. The relationship between the rules is illustrated graphically.

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

2 XML Schema design principles

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

2.1 Notes on UML2.0 semantics of the AUTOSAR meta-model

In UML2.0 [5] attributes and navigable association ends are represented as properties. Since the AUTOSAR Template Profile only supports associations with two association ends, attributes and associations can be considered as equivalent for the XML schema production rules. Therefore the XML schema production rules 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 meta-model (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 [5].

In the following sections the value of the attribute "aggregation" of the navigable association end is shown in brackets behind the association.

In the following sections the term "Property" will always mean a primitive attribute of a class or the target end of an association that is either a reference (aggregation = none) or a composition (aggregation = composite).

2.1.1 Representation of association (aggregation = composite)

Figure 2.1 depicts how a composite association is represented by means of instances of the UML2.0 meta-model. 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 XML schema production rules: From the point of view of production rules there is not difference between composite association and an attribute (see also next section).

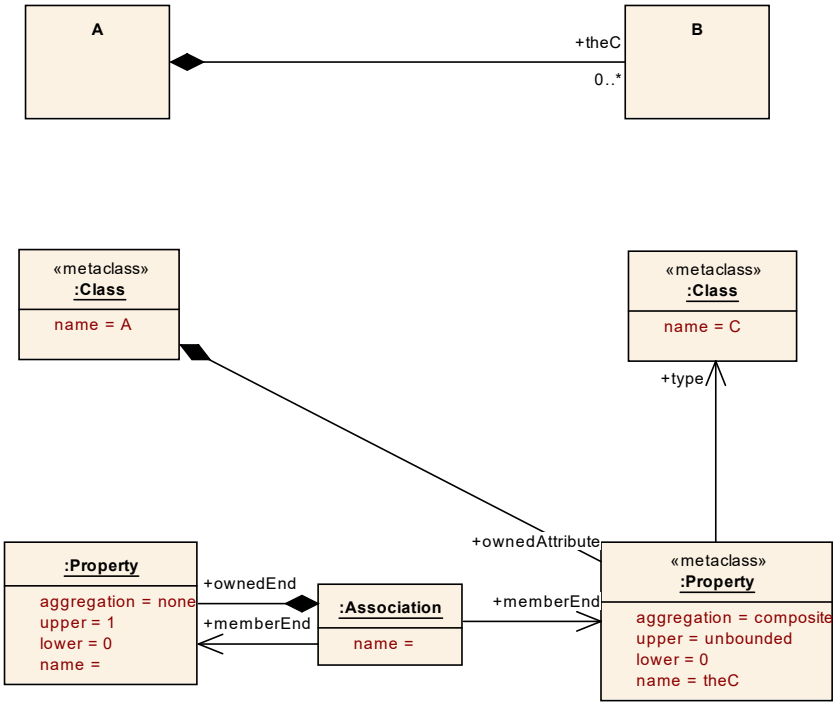

Figure 2.1: Representation of association (aggregation = composite)

2.1.2 Representation of attribute (aggregation = composite)

Figure 2.2 depicts how an attribute is represented by means of instances of the UML2.0 meta-model. The attribute 'theC' is represented as a property that is owned by class 'A'.

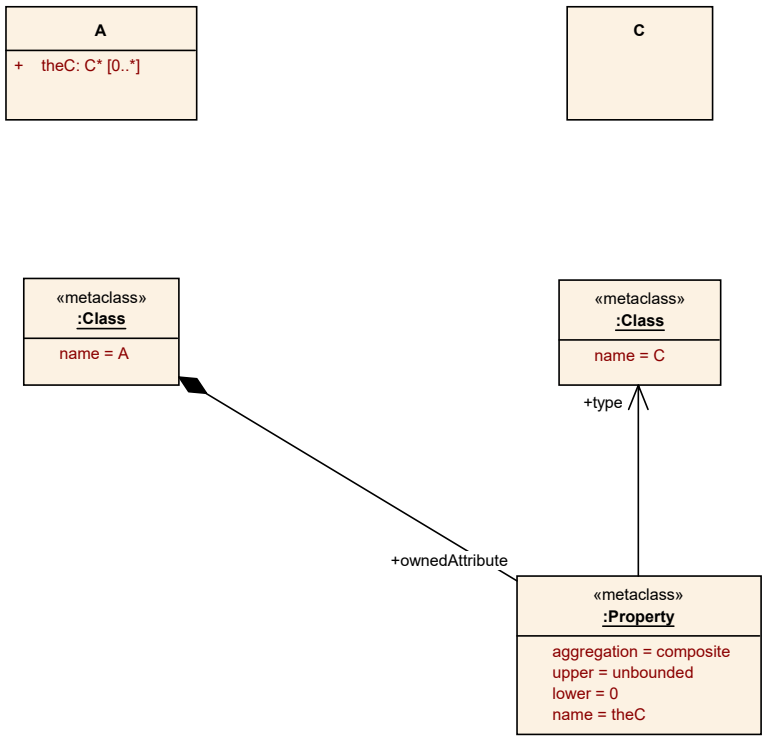

Figure 2.2: Representation of attribute (aggregation = composite)

2.1.3 Representation of associations (aggregation = none)

Figure 2.3 depicts how a reference (association with aggregation = none) is represented by means of instances of the UML2.0 meta-model. 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 XML schema production rules. From the point of view of the production rules *there is no difference between a reference and an attribute with aggregation=none* (see also next section). However, the AUTOSAR meta-model allows stereotypes for references. The special semantics are handled separately as described in Chapter 3.2.3.

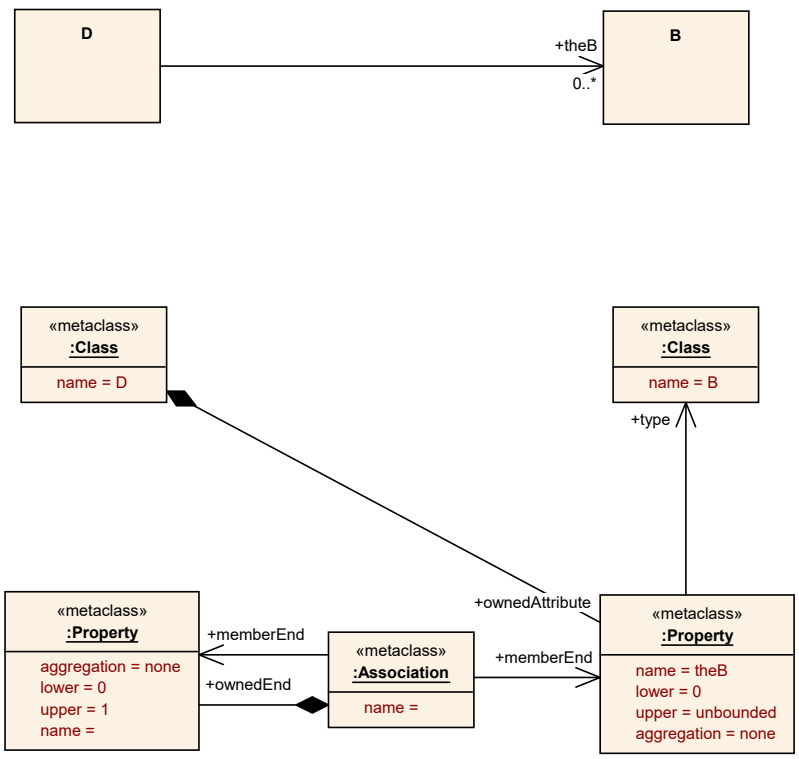

Figure 2.3: Representation of association (aggregation = none)

2.1.4 Representation of attributes (aggregation = none)

Figure 2.4 depicts how an attribute with aggregation = none is represented by means of instances of the UML2.0 meta-model. 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 only attributes with aggregation=composite are allowed. However, the XML schema production rules cover those attributes since they do not add complexity: For the XML schema production rules attributes with aggregation=none (described in this section) are equivalent to associations with aggregation=none (described in Chapter 2.1.3).

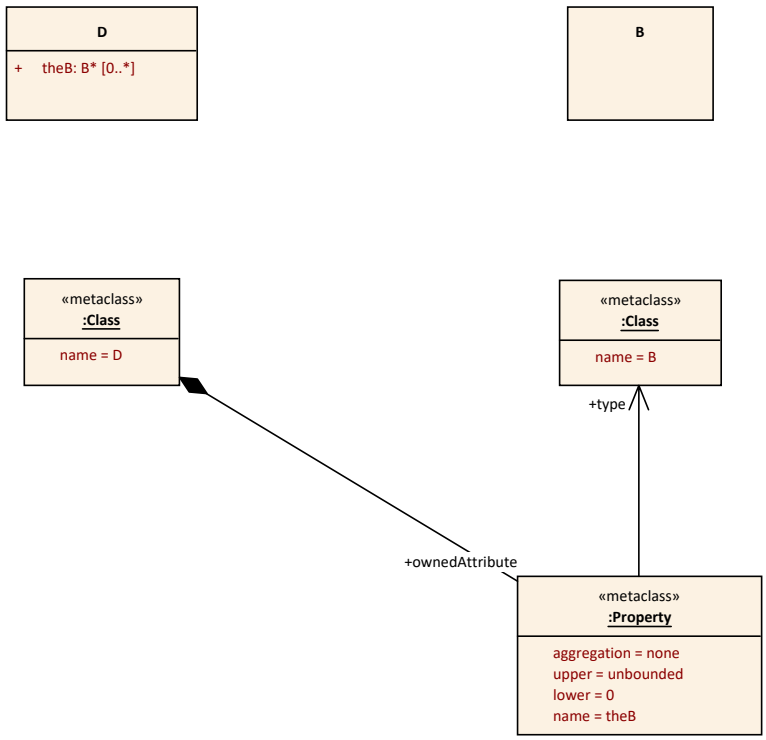

Figure 2.4: Representation of attribute (aggregation = none)

2.2 Notes on use of W3C XML schema

A W3C XML schema provides 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.

[TPS_XMLSPR_00054] W3C XML Schema Version 1.0 ⌈The AUTOSAR XML Schema conforms to the W3C XML Schema Version 1.0 (see [1]).⌋

The production rules described in this document make sure that for each class, attribute and association represented in the AUTOSAR meta-model a representation in the AUTOSAR XML schema exists. Additionally, they make sure that the mapping between the AUTOSAR meta-model and the AUTOSAR schema is unambiguous.

In other words:

- An instance of the AUTOSAR meta-model 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.

2.3 Handling Inheritance

[TPS_XMLSPR_00029] Inheritance in the AUTOSAR XML Schema ⌈Inheritance in the AUTOSAR meta-model is mapped to XML schema by the following mechanisms:

- For each class in the AUTOSAR meta-model groups are created which contain the XML-elements and XML-attributes that represent the properties that are directly defined by the class. Classes without properties with xml.attribute=false will result in empty groups.
- Additionally an xsd:complexType representing the full content of the concrete class is created. The structure of this xsd:complexType is defined by referencing the group that defines the properties of the class itself and the xsd:groups that define the properties of the super-classes. The group 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 Chapter 2.7 for more details on the order of XML-elements and groups).

2.4 Generic Approach

The AUTOSAR XML schema production 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 [8] and XHTML [9].

2.5 XML element versus attribute

In accordance to the MSR-TR-CAP the production 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 3.1.2.1 for more details on this tagged value).

2.6 XML names

[TPS_XMLSPR_00030] XML Names ⌈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 parts of the names.

This document refers to a name that is translated as described in this section as a **XML-name**.

Non-alphanumeric characters SHALL not be used in UML names.

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

1. Remove all non-alphanumeric characters from the UML name. If such characters exist, raise an error.
2. Split up the UML name from left to right into tokens. A new token starts whenever an uppercase letter or digit is found.
   TestECUClass12ADC -> [Test, E, C, U, Class, 1, 2, A, D, C]
3. 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]
4. Convert all tokens to uppercase:
   E.g.: [TEST, ECU, CLASS, 12, ADC]
5. Concatenate 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. In this case, the value of xml.name SHALL NOT be empty, only alphanumeric characters and hyphens SHALL be used as the value of xml.name, and the first and the last character of the value SHALL be alphanumeric.

[TPS_XMLSPR_00031] XML Name Plurals ⌈A plural XML-name is created by appending an "S" to the singular XML-name.⌋

If 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 meta-model names to names used in the XML schema:

Name in AUTOSAR meta-model	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

Table 2.1: Examples for mapping of AUTOSAR meta-model names to XML names

2.7 Order of XML-elements

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.

2.7.1 Order of xml elements

[TPS_XMLSPR_00032] Order of XML elements ⌈Properties owned by classes in the AUTOSAR meta-model are mapped to XML-elements. By default, the AUTOSAR XML schema defines a certain sequence on XML elements which follows an alphabetical ordering[1].⌋

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

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:

$$offsetA < offsetB \Rightarrow \texttt{elementA} \text{ is listed before } \texttt{elementB}$$

Elements with the same offset are ordered alphabetically.

2.7.2 Order of xml elements of derived uml:properties

Chapter 2.7.1 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.

[TPS_XMLSPR_00033] Order of XML elements with Inheritance ⌈The XML elements that represent XML elements directly owned by a class are grouped together and ordered as described in [TPS_XMLSPR_00032]. 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
    for (int i=0; i<classesWithoutSupertypeIdentifiable ; 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 2.5 shows an example of the ordering of XML elements within the XML schema. The numbers next to class `Identifiable` indicate the sequenceOffset of directly owned properties. The comments indicate the sequence of the groups of XML elements.

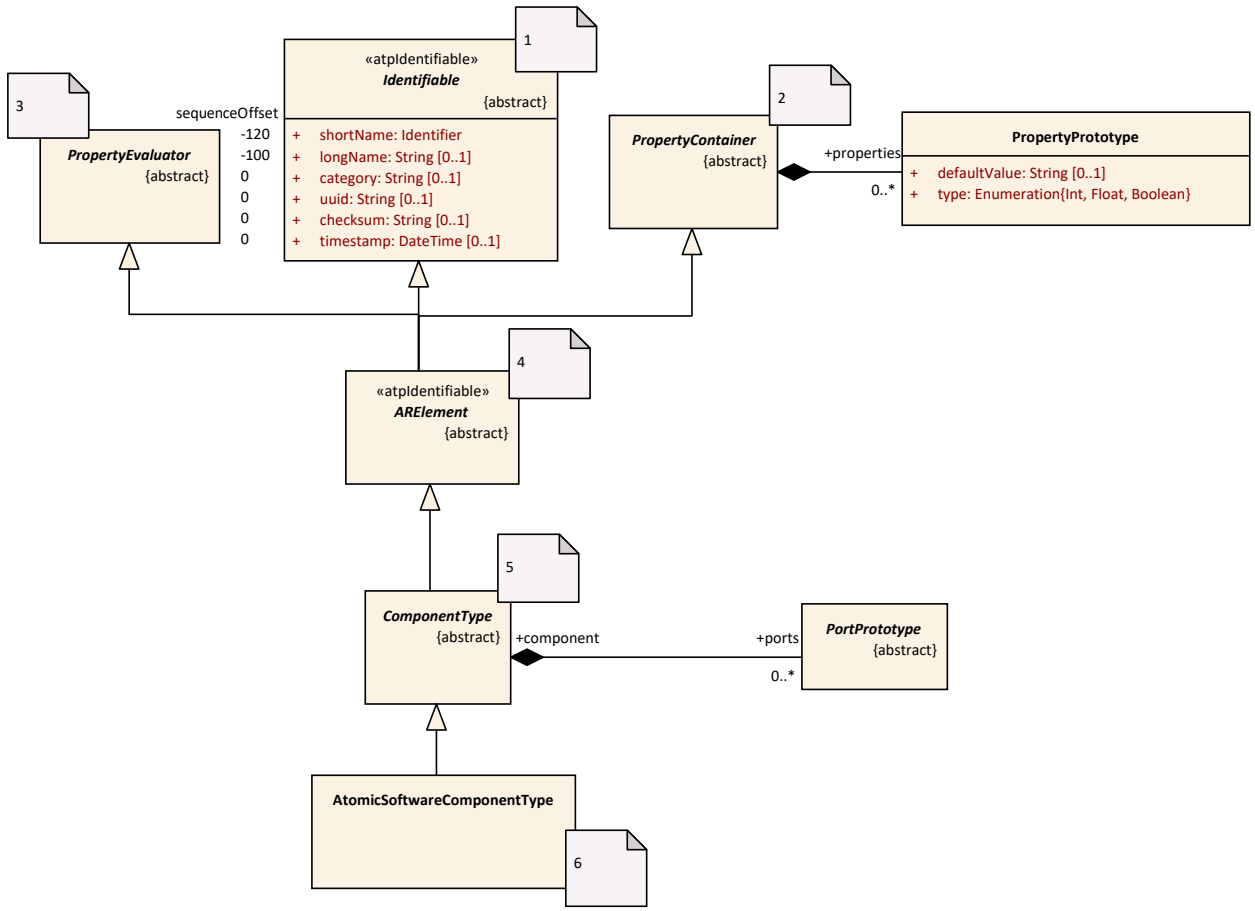

Figure 2.5: Order of XML elements

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

An xsd:group is created for all classes. The XML elements in each xsd:group are ordered as defined in section 2.7.1 (in this example all properties are mapped to XML elements):

- xsd:group for `Identifiable`:

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

  SHORT-NAME is listed first because it has the smallest sequenceOffset (-120). It is followed by LONG-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"/>
    <xsd:group ref="AR:PROPERTY-EVALUATOR"/>
    <xsd:group ref="AR:AR-ELEMENT"/>
    <xsd:group ref="AR:COMPONENT-TYPE"/>
    <xsd:group ref="AR:ATOMIC-SOFTWARE-COMPONENT-TYPE"/>
  </xsd:sequence>
</xsd:complexType>
```

2.8 Linking

[TPS_XMLSPR_00034] XML Representation of meta-class references ⌈References between meta-classes are represented by XML-elements suffixed with <...-REF> or <...-TREF>.⌋

For more details about the representation of associations in XML descriptions, please refer to Generic Structure Template [10].

2.9 Transmitting incomplete Data

[TPS_XMLSPR_00028] Optional XML elements ⌈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 meta-model 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.

2.10 Identification of XML schema version in XML descriptions

[TPS_XMLSPR_00035] XML schema version ⌈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/schema/r<major>.<minor>.⌋

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 4.0 is http://autosar.org/schema/r4.0.

3 Configuration of XML schema production

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

- **Step1: Configuration of XML schema production.**
  In a first step the AUTOSAR meta-model is translated to a simplified intermediate model[1]. Content relevant information of the AUTOSAR meta-model is mapped to classes, properties, enumerations, primitive data types and tagged values. If not otherwise mentioned missing tagged values are set to their default value as described in section 3.1.1ff. Some more complex mappings are described in chapter 3.2.
- **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 4 for more details on the schema production rules.

3.1 Tailoring schema production

3.1.1 Overview

[TPS_XMLSPR_00036] Tailoring XML schema production ⌈

Tag Name	Value Type	Default Value	Applica-ble to	Description
extension-Point	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.
xml.attribute	Boolean	False	Property: Attributes only	If true, serializes the property as an XML attribute. By default all properties are mapped to XML elements.

▽

[1] The intermediate model only uses concepts which are available in EMOF [11].

△

Tag Name	Value Type	Default Value	Applica-ble to	Description
xml. attributeRef	Boolean	False	Property	If true, serializes the property as an XML attribute reference (e.g. <xsd:attribute ref="xml:space" >). The namespace is taken from the value of xml.nsPrefix, the value of xml.name must be a valid name in that namespace.
xml. enforceMax-Multiplicity	Boolean	True	Property	If true, enforce maximum multiplicity; otherwise, it is "unbounded". By default xml.enforceMaxMultiplicity is true.
xml.enforce Min Multiplicity	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.glob-alElement	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 meta-model. Usually only the meta-class AUTOSAR is represented by a globally defined XML element.
xml.mds.type	String	Empty	Class with stereotype «primitive»	This tag identifies the xsd:simpleType or xsd:complexType which represents the primitive data type in the meta-model. In contrast to the 'xml.xsd.customType' tag, this tagged value indicates a schema fragment that is predefined within the schema generator. Therefore types tagged by 'xml.mds.type' are not generated into the schema based on the tagged class within the model.

▽

▽

△

Tag Name	Value Type	Default Value	Applicable to	Description
				△ Hence using this tag requires knowledge of the schema generation process it has to be ensured that only types created within the generation process are referenced by this tag.
xml.name	String	See column "Description"	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 meta-model, then this value is calculated as explained in [TPS_XMLSPR_00030].
xml. namePlural	String	See column "Description"	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 meta-model, then this value is calculated as explained in [TPS_XMLSPR_00031].
xml.nsPrefix	String	AR	Package, Class, Property	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.
xml.nsURI	String	http://autosar.org/schema/r<major>.<minor>	Package, Class	By default all XML-elements are assigned to the namespace http://autosar.org/schema/r<major>.<minor> 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.ordered	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.
xml.roleElement	Boolean	See column "Description"	Property	If set to true, the xml.name of the role shows up as an XML-element. If not explicitly defined in the AUTOSAR meta-model, then the default rules as described in [TPS_XMLSPR_00056] are applied.
xml.roleWrapperElement	Boolean	See column "Description"	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 meta-model, then the default rules as described in [TPS_XMLSPR_00056] are applied.

▽

△

Tag Name	Value Type	Default Value	Applicable to	Description
xml.sequenceOffset	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 meta-model, then the xml.sequenceOffset is set to 0.
xml.text	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.typeElement	Boolean	See column "Description"	Property	If set to true, the xml.name of the type shows up as an XML-element. If not explicitly defined in the AUTOSAR meta-model, then the default rules as described in [TPS_XMLSPR_00056] are applied.
xml.typeWrapperElement	Boolean	false, see also column "Description"	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 [TPS_XMLSPR_00056] are applied.
xml.xsd.customType	String	Empty	Class with stereotype «primitive»	This tag is applicable to a «primitive». It specifies the name of the xsd:simpleType which represents the primitive type.

▽

△

Tag Name	Value Type	Default Value	Applicable to	Description
xml.xsd.maxLength	Integer	Empty	Class with stereotype «primitive»	Length restriction for defining a custom primitive type based on a string type. xml.xsd.type must be string.
xml.xsd.pattern	String	Empty	Class with stereotype «primitive»	Regular expression, used as restriction for defining a custom primitive type based on a string type. xml.xsd.type must be string.
xml.xsd.type	String	Empty	Class with stereotype «primitive»	This tag identifies the xsd:simpleType which represents the primitive data type in the meta-model. The value refers to a W3C XML schema data type. 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.xsd.whitespace	String	Empty	Class with stereotype «primitive»	Flag, whether whitespace in the value of the primitive type needs to be preserved. Valid values are {preserve, default}.

Tagged values that can be used for tailoring the mapping rules

⌋

3.1.2 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.

3.1.2.1 Constraints on tags applied to properties

[TPS_XMLSPR_00057] 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 an 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 data type.

3.1.2.2 Constraints on tags applied to classes

[TPS_XMLSPR_00058] Constraints on tags applied to classes ⌈

Constraints 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 the Generic Structure Template [10]. If these tagged values are applied, then all other tagged values are ignored.

3.1.2.3 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.

[TPS_XMLSPR_00059] Constraints on values of xml.*Element tagged values ⌈

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 multiplicity = 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 multiplicity = 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	Handle with care: Type Element and Role Information is missing - might lead to ambiguous descriptions if used with upper multiplicity > 1 and/or inheritance. It is a kind of shortchut for polymorphism and applies to very specific usecases.	H	

▽

△

xml.roleWrapperElement	xml.roleElement	xml.typeWrapperElement	xml.typeElement	description	Usage (P=preferred, A=alternative, H=handle with care, N=not allowed)	used in standard
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

⌋

3.2 Default configuration of XML schema production

This chapter describes how the XML schema production rules are configured for mapping the AUTOSAR meta-model to the AUTOSAR XML schema. Tagged values that are already defined in the AUTOSAR meta-model 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 meta-model.

3.2.1 Configuration of multiplicities

[TPS_XMLSPR_00037] XML Configuration of multiplicities ⌈The tagged values 'xml.enforceMinMultiplicity' and 'xml.enforceMaxMultiplicity' are set to the default values (see [TPS_XMLSPR_00036]) 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.

⌋

3.2.2 Mapping configuration for properties

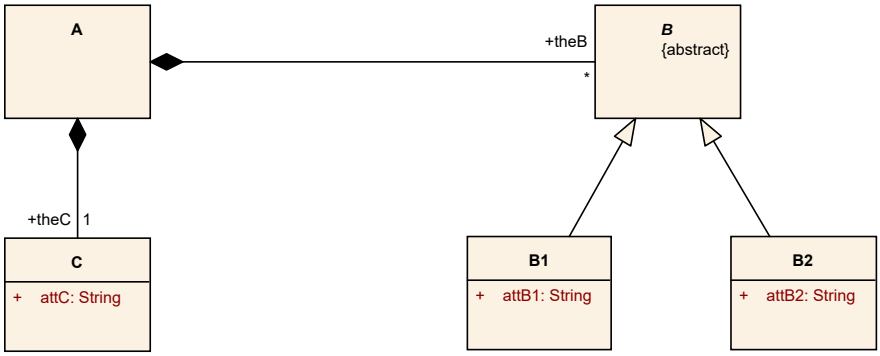

Figure 3.1: Example meta-model

Five cases are distinguished when configuring the mapping of properties in the AUTOSAR meta-model:

1. **[TPS_XMLSPR_00038] XML Configuration of properties (upper multiplicity 1, no subclasses)** ⌈Upper multiplicity of property = 1 and type of property has no subclasses:
   - 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 meta-model

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.⌋

The property 'theC' in Figure 3.1 depicts this case.

2. **[TPS_XMLSPR_00039] XML Configuration of properties (upper multiplicity greater than 1, no subclasses)** ⌈Upper multiplicity of property > 1 and type of property has no subclasses:

   - 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. **[TPS_XMLSPR_00040] XML Configuration of properties (upper multiplicity 1, with subclasses)** ⌈Upper multiplicity of property = 1 and type of property has subclasses:

   - 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 meta-model.

   ⌋

4. **[TPS_XMLSPR_00041] XML Configuration of properties (upper multiplicity greater than 1, with subclasses)** ⌈Upper multiplicity of property > 1 and type of property has subclasses:

   - 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 meta-model.

   ⌋

   The property 'theB' in Figure 3.1 depicts this case.

5. **[TPS_XMLSPR_00042] XML Configuration of properties (upper multiplicity greater than 1, primitive type or enum or association)** ⌈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 meta-model are not overwritten. Therefore the mapping to XML can individually be configured if the default mappings are not sufficient.

[TPS_XMLSPR_00056] Default settings for the XML Configuration of properties

⌈

	Case			Default configuration of Schema						
applicable for AUTOSAR	Upper Multiplicity	Target is Abstract Class	Kind of Property	xml.roleWrapperElement	xml.roleElement	xml.typeWrapperElement	xml.typeElement	Wrapper name (usually xml. namePlural)	Inner name (usually xml.name)	Comment
yes	=1	no	Composition	0	1	0	0		{role}	
yes	=1	no	Reference	0	1	0	0		{role}Ref	
yes	=1	no	Concrete InstanceRef	0	1	0	0		{role}Iref	
yes	=1	no	Abstract InstanceRef	0	1	0	1		{role}Iref	Subclassing of InstanceRef is represented by typeElement
yes	=1	no	IsOfType	0	1	0	0		{role}Tref	
yes	=1	no	Attribute	0	1	0	0		{role}	
yes	=1	yes	Composition	0	1	0	1	{role}	{type}	
yes	=1	yes	Reference	0	1	0	0		{role}Ref	Subclassing is represented in dest
yes	=1	yes	Concrete InstanceRef	0	1	0	0		{role}Iref	Subclassing is represented in dest of atpTarget
yes	=1	yes	Abstract InstanceRef	0	1	0	1		{role}Iref	Subclassing of target is represented in dest of atpTarget; Subclassing of InstanceRef is represented by typeElement
yes	=1	yes	IsOfType	0	1	0	0		{role}Tref	Subclassing is represented in dest
no	=1	yes	Attribute	0	1	0	0		{role}	
yes	>1	no	Composition	1	0	0	1	{role}s	{type}	
yes	>1	no	Reference	1	1	0	0	{role}Refs	{role}Ref	
yes	>1	no	Concrete InstanceRef	1	1	0	0	{role}Irefs	{role}Iref	
yes	>1	no	Abstract InstanceRef	1	1	0	1	{role}Irefs	{role}Iref	Subclassing of InstanceRef is represented by typeElement
no	>1	no	IsOfType	1	1	0	0	{role}Trefs	{role}Tref	
yes	>1	no	Attribute	1	1	0	0	{role}s	{role}	
yes	>1	yes	Composition	1	0	0	1	{role}s	{type}	
yes	>1	yes	Reference	1	1	0	0	{role}Refs	{role}Ref	Subclassing is represented in dest
yes	>1	yes	Concrete InstanceRef	1	1	0	0	{role}Irefs	{role}Iref	Subclassing is represented in dest of atpTarget

▽

△

	Case			Default configuration of Schema						
applicable for AUTOSAR	Upper Multiplicity	Target is Abstract Class	Kind of Property	xml.roleWrapperElement	xml.roleElement	xml.typeWrapperElement	xml.typeElement	Wrapper name (usually xml. namePlural)	Inner name (usually xml.name)	Comment
yes	>1	yes	Abstract InstanceRef	1	1	0	1	{role}Irefs	{role}Iref	Subclassing of target is represented in dest of atpTarget; Subclassing of InstanceRef is represented by typeElement
no	>1	yes	IsOfType	1	1	0	0	{role}Trefs	{role}Tref	Subclassing is represented in dest
no	>1	yes	Attribute	1	1	0	0	{role}s	{role}	

The default settings for the XML Configuration of properties in production of the AUTOSAR XML schema

⌋

3.2.3 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).

3.2.3.1 References without stereotypes

[TPS_XMLSPR_00043] XML Configuration of references without stereotype ⌈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 '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 [TPS_XMLSPR_00030]). The 'xml.namePlural' is set to the default 'xml.name' appended by "-REFS".⌋

3.2.3.2 Instance references

The AUTOSAR Template UML Profile requires that all details of instance references are properly modeled in the AUTOSAR meta-model.

[TPS_XMLSPR_00044] XML Configuration of instance references ⌈The following tagged values are applied:

- Composite reference between the source of the instance reference and the meta-class 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.roleWrapperElement = [false if upper multiplicity of reference = 1; true if upper multiplicity of reference > 1]
  - xml.roleElement = true
  - xml.typeWrapperElement = false
  - xml.typeElement = [false if InstanceRef class is concrete; true if InstanceRef class is abstract] (If there are subclasses of the InstanceRef class, the applied class has to be specified by a typeElement.)
- instanceRef meta-class:
  - 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 meta-class to the target (the reference to the target is mandatory):
  - xml.enforceMinMultiplicity = false (Consequently, this should be set to true as an InstanceRef shall always point to a target. However, not to break backward-compatibility this has to remain false.)
  - xml.enforceMaxMultiplicity = true
  - xml.sequenceOffset = 9999 (the target is the last element in the list of references)
- references from instanceRef meta-class 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

⌋

Example

Figure 3.2 shows an example of a detailed representation of an instance reference in the AUTOSAR meta-model.

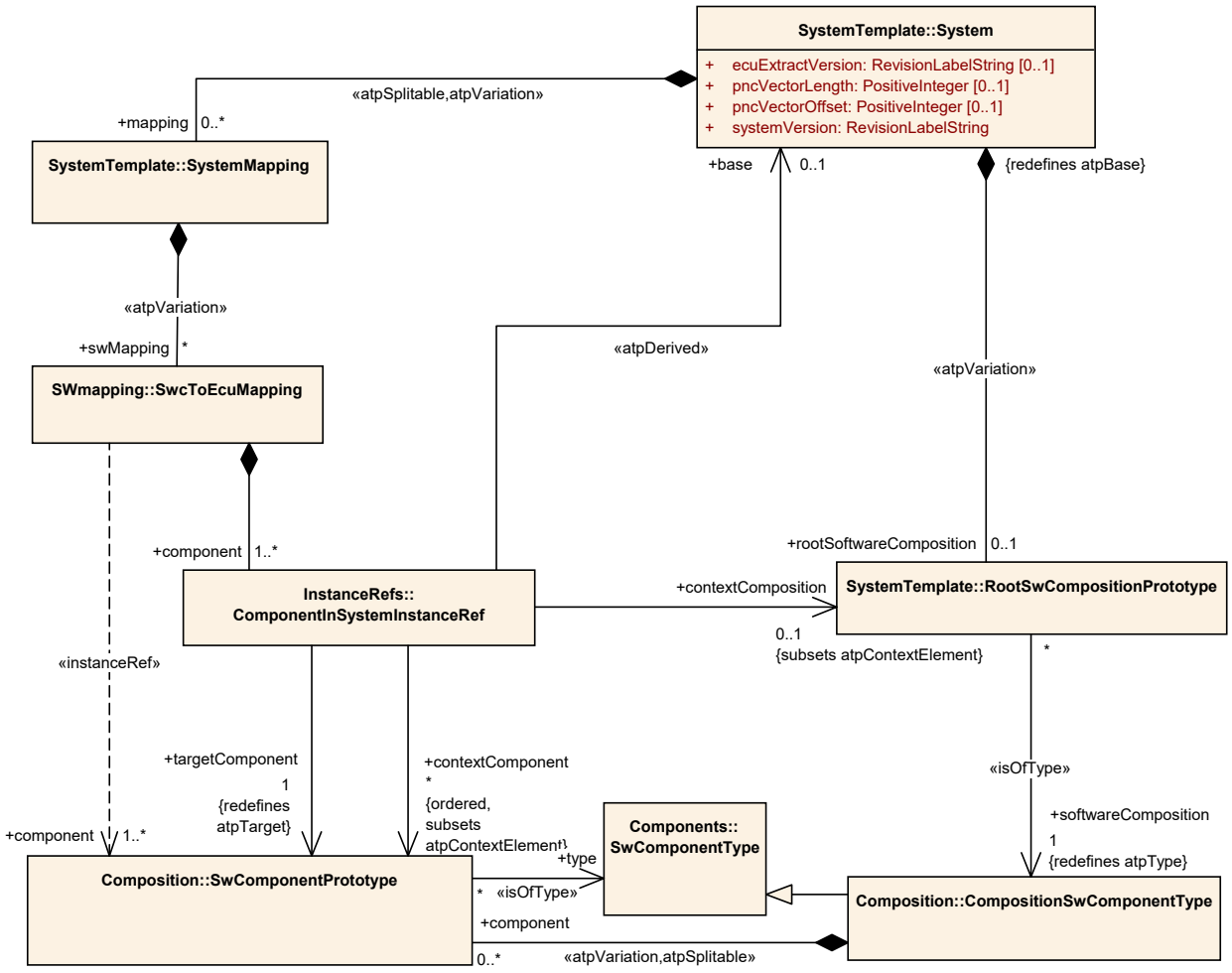

Figure 3.2: Example of an instanceRef association

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

```
<!-- complex type for class InstanceRefs::ComponentInCompositionInstanceRef
     -->
<xsd:complexType name="COMPONENT-IN-COMPOSITION-INSTANCE-REF" abstract="
    false" mixed="false">
  <xsd:annotation>
    <xsd:documentation>The ComponentInCompositionInstanceRef points to a
       concrete SwComponentPrototype within a CompositionSwComponentType.</
       xsd:documentation>
    <xsd:appinfo source="tags">mmt.qualifiedName="
       ComponentInCompositionInstanceRef"</xsd:appinfo>
    <xsd:appinfo source="stereotypes">atpObject,instanceRef</xsd:appinfo>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:group ref="AR:AR-OBJECT"/>
    <xsd:group ref="AR:ATP-INSTANCE-REF"/>
    <xsd:group ref="AR:COMPONENT-IN-COMPOSITION-INSTANCE-REF"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="AR:AR-OBJECT"/>
</xsd:complexType>
<!-- element group for class InstanceRefs::ComponentInSystemInstanceRef -->
<xsd:group name="COMPONENT-IN-SYSTEM-INSTANCE-REF">
  <xsd:annotation>
    <xsd:appinfo source="tags">mmt.qualifiedName="
       ComponentInSystemInstanceRef"</xsd:appinfo>
    <xsd:appinfo source="stereotypes">atpObject,instanceRef</xsd:appinfo>
  </xsd:annotation>
  <xsd:sequence>
  <!-- Association <<atpDerived>>base skipped -->
    <xsd:element name="CONTEXT-COMPOSITION-REF" minOccurs="0">
      <xsd:annotation>
        <xsd:appinfo source="tags">mmt.qualifiedName="
           ComponentInSystemInstanceRef.contextComposition";pureMM.
           maxOccurs="1";pureMM.minOccurs="0";xml.sequenceOffset="20"</
           xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="AR:REF">
            <xsd:attribute name="DEST" type="AR:ROOT-SW-COMPOSITION-
               PROTOTYPE--SUBTYPES-ENUM" use="required"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="CONTEXT-COMPONENT-REF" minOccurs="0" maxOccurs="
       unbounded">
      <xsd:annotation>
        <xsd:appinfo source="tags">mmt.qualifiedName="
           ComponentInSystemInstanceRef.contextComponent";pureMM.maxOccurs=
           "-1";pureMM.minOccurs="0";xml.sequenceOffset="30"</xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="AR:REF">
            <xsd:attribute name="DEST" type="AR:SW-COMPONENT-PROTOTYPE--
               SUBTYPES-ENUM" use="required"/>
```

```
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="TARGET-COMPONENT-REF" minOccurs="0">
      <xsd:annotation>
        <xsd:appinfo source="tags">mmt.qualifiedName="
            ComponentInSystemInstanceRef.targetComponent";pureMM.maxOccurs="
            1";pureMM.minOccurs="1";xml.sequenceOffset="40"</xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="AR:REF">
            <xsd:attribute name="DEST" type="AR:SW-COMPONENT-PROTOTYPE--
                SUBTYPES-ENUM" use="required"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:group>
```

An example instanceRef looks like:

```
<...>
  <COMPONENT-IREFS>
    <COMPONENT-IREF>
      <CONTEXT-COMPOSITION-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">/
          theSystem/aVehicle</CONTEXT-COMPOSITION-REF>
      <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/vendor/
          theEngine</CONTEXT-COMPONENT-REF>
      <TARGET-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/theEngineVendor/
          engineSpeedDetermination</TARGET-COMPONENT-REF>
    </COMPONENT-IREF>
  </COMPONENT-IREFS>
</...>
```

3.2.3.3 References with stereotype ≪isOfType≫

[TPS_XMLSPR_00045] XML Configuration of type references ⌈If the stereotype ≪isOfType≫ is applied to an association in the AUTOSAR meta-model then the tagged value 'xml.name' of the navigable association end is set to the default xml.name appended by "-TREF". According to the AUTOSAR Template Profile, 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.⌋

3.2.4 Stereotypes applied to classes

3.2.4.1 Stereotype ≪atpMixed≫

If the stereotype ≪atpMixed≫ is applied to a class in the AUTOSAR meta-model then the properties are represented by XML elements in arbitrary order and unbounded multiplicity. This only applies to properties that are not explicitly mapped to attributes by setting the 'xml.attribute' tag to 'true'.

[TPS_XMLSPR_00046] XML Configuration of classes with ≪atpMixed≫ ⌈The following values are applied to classes with stereotype ≪atpMixed≫ (other values specified in the meta-model are ignored):

- Values of the stereotyped meta-class:
  - xml.ordered=false
  - xml.text=false
- Values of the properties of the stereotyped meta-class:
  - 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)

⌋

3.2.4.2 Stereotype ≪atpMixedString≫

If the stereotype ≪atpMixedString≫ is applied to a class in the AUTOSAR meta-model then the properties may be represented by XML elements in arbitrary order and unbounded multiplicity. In this case the tagged value 'xml.ordered' is set to false and the tagged value 'xml.text' is set to true. See chapter 3.1.1 for more details on the scope of the tagged value 'xml.text'. No wrappers are created for the properties. Additionally, the XML elements may have text in-between.

[TPS_XMLSPR_00047] XML Configuration of classes with ≪atpMixedString≫ ⌈The following values are applied to classes with stereotype ≪atpMixedString≫ (other values specified in the meta-model are ignored):

- Values of the stereotyped meta-class:
  - xml.ordered=false

  - xml.text=true

- Values of the properties of the stereotyped meta-class:
  - 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 XML Schema production rules

The following sections describe the mapping rules for an automatic generation of the AUTOSAR XML schema out of the intermediate meta-model. Please note that in the intermediate meta-model all tagged values and multiplicities are set as defined in chapter 3.

Each rule is described by the following information:

- **Applies to:**
  The meta-meta-model (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 meta-model element is mapped to XML schema. Values that need to be read out of the AUTOSAR meta-model are denoted by script tags "<%" and "%>":
  e.g.: <%=my variable %>.
- **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.

4.1 Create model representation

Figure 4.1 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.

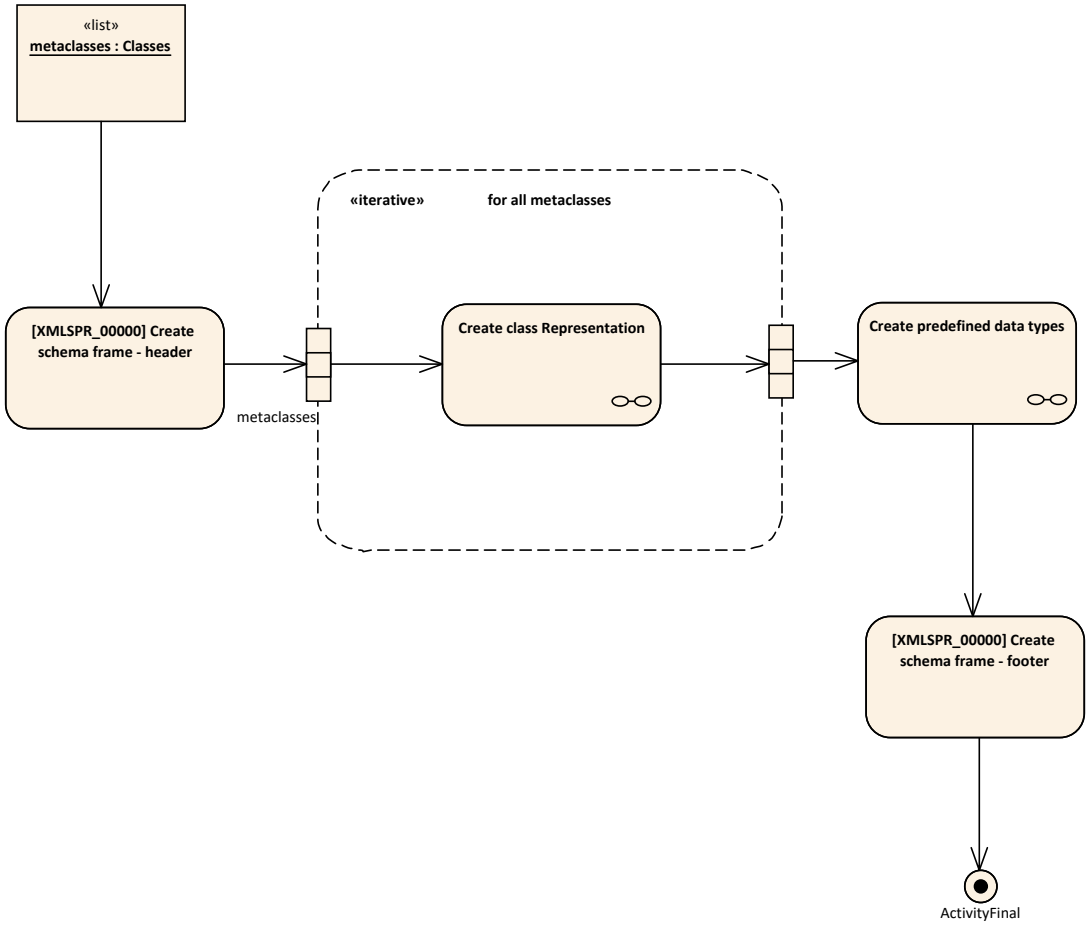

Figure 4.1: Model representation

4.1.1 Create xsd:schema

[TPS_XMLSPR_00000] XML Schema production rule: 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">`
	`<xsd:import namespace="http://www.w3.org/XML/1998/namespace"`
	`schemaLocation="http://www.w3.org/2001/03/xml.xsd"/>`
	
	`<% for all classes { %>`
	`<% call class representation %>`
	`<% } %>`
	`</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://autosar.org/schema/r<realease_number>. The body of the XML schema is composed of a namespace import of the XML namespace and representations for each class (including their properties).

XML schema example:	`<xsd:schema xmlns:AR="http://autosar.org/schema/r4.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://autosar.org/schema/r4.0" elementFormDefault="qualified" attributeFormDefault="unqualified" >`
	`...`
	`</xsd:schema>`

XML instance example:	`<... xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar.xsd">...` `</...>`

⌋

4.2 Create class representation

Figure 4.2 depicts XML schema fragments created for each class:

- If the stereotype ≪enumeration≫ is applied, then the class is mapped to an xsd:enumeration [TPS_XMLSPR_00007].
- If the stereotype ≪primitive≫ is applied, then the data type denoted by the tagged value 'xml.mds.type' or 'xml.xsd.type' are used to represent the class within the schema [TPS_XMLSPR_00006].
- Otherwise:
  - If the class owns properties with 'xml.attribute=true' then an xsd:attributeGroup is created [TPS_XMLSPR_00002].
  - Additionally if the class owns properties with 'xml.attribute=false' then an xsd:group is created [TPS_XMLSPR_00001].
  - Additionally if the class is not abstract then an xsd:complexType is created [TPS_XMLSPR_00003]. If the tagged value 'xml.globalElement' is set to true, then a global XML element declaration is created.
  - Additionally if the uml:class is referenced then an xsd:simpleType that represents the lists possible concrete instances of the uml:class [TPS_XMLSPR_00025].

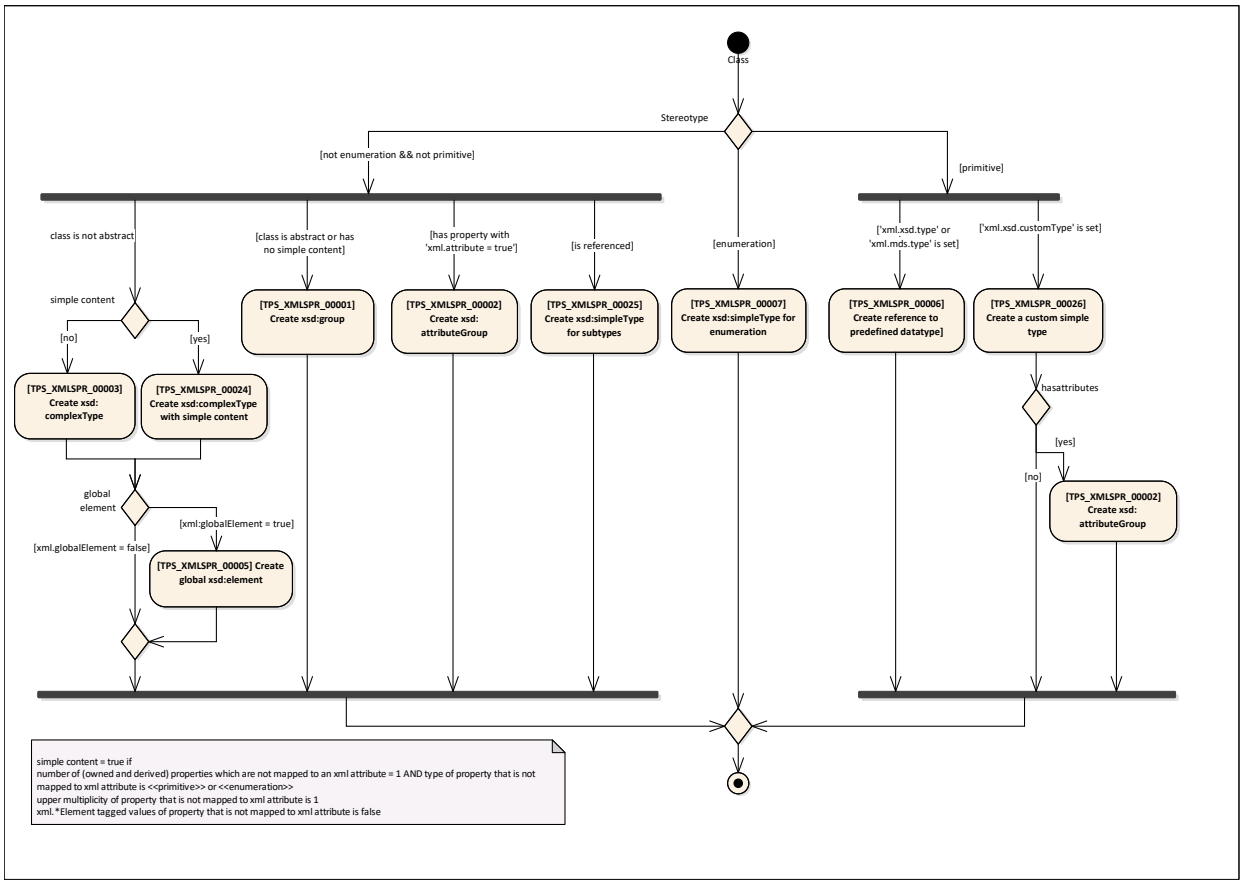

Figure 4.2: Class representation

4.2.1 Create xsd:group

[TPS_XMLSPR_00001] XML Schema production rule: xsd:group ⌈

Applies to	Class without stereotype ≪enumeration≫ or stereotype ≪primitive≫
Precondition	NOT (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	<pre><xsd:group name="<%=xmlName%>">
	  <% if (ordered) { %>
	  <xsd:sequence>
	  <% } else { %>
	  <xsd:choice>
	  <xsd:choice maxOccurs="unbounded" minOccurs="0">
	  <% } // end if %>
	  <% for all properties {
	  <% call rule
	  TPS_XMLSPR_00008 |
	  TPS_XMLSPR_00009 |
	  TPS_XMLSPR_00023 |
	  TPS_XMLSPR_00022 |
	  TPS_XMLSPR_00010 |
	  TPS_XMLSPR_00011 |
	  TPS_XMLSPR_00012 |
	  TPS_XMLSPR_00013 |
	  TPS_XMLSPR_00014 |
	  TPS_XMLSPR_00015 |
	  TPS_XMLSPR_00016 |
	  TPS_XMLSPR_00055 %>
	  } // end for %>
	  <% if (ordered) { %>
	  </xsd:sequence>
	  <% } else { %>
	  </xsd:choice>
	  </xsd:choice>
	  <% } // end if %>
	</xsd:group></pre>

Description:	If the class is abstract or owns at least one property with '**xml.attribute=false**' the condition for an **xsd:complexType with simple content** is not met (see [TPS_XMLSPR_00024]), 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 [TPS_XMLSPR_00032]. If the tagged value '**xml.ordered=true**' is set then the contents are listed in an xsd:sequence. Otherwise they are listed within an xsd:choice. If there are no child elements of the class (i.e. all children have '**xml.attribute=false**' or are ≪`atpAbstract`≫ or ≪`atpDerived`≫), an empty xsd:sequence is generated. The XML-elements representing the properties are created by rules [TPS_XMLSPR_00008], [TPS_XMLSPR_00009], [TPS_XMLSPR_00023], [TPS_XMLSPR_00022], [TPS_XMLSPR_00010], [TPS_XMLSPR_00011], [TPS_XMLSPR_00012], [TPS_XMLSPR_00013], [TPS_XMLSPR_00014], [TPS_XMLSPR_00015], [TPS_XMLSPR_00016], [TPS_XMLSPR_00055]

XML schema example:	<pre><xsd:group name="IDENTIFIABLE">
	  <xsd:sequence>
	  <!-- property representations created by rules TPS_XMLSPR_00008,
	  TPS_XMLSPR_00009, TPS_XMLSPR_00023, TPS_XMLSPR_00022,
	  TPS_XMLSPR_00010, TPS_XMLSPR_00011, TPS_XMLSPR_00012,
	  TPS_XMLSPR_00013, TPS_XMLSPR_00014, TPS_XMLSPR_00015,
	  TPS_XMLSPR_00016, TPS_XMLSPR_00055 -->
	  <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></pre>

▽

△

XML instance example:	`<...>`
	`<SHORT-NAME>theShortName</SHORT-NAME>`
	`<LONG-NAME>theLongtName</LONG-NAME>`
	`<CATEGORY>theCategory</ CATEGORY >`
	`</...>`

⌋

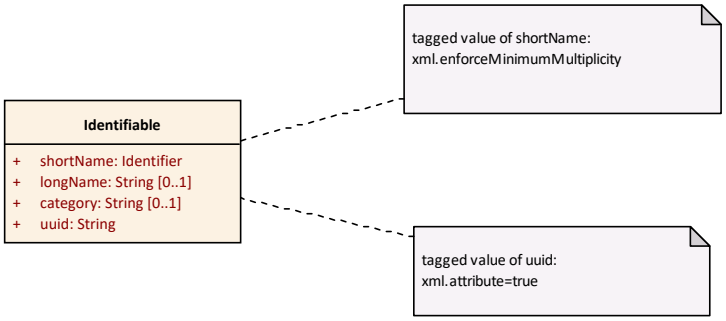

Figure 4.3: UML example - Create xsd:group

4.2.2 Create xsd:attributeGroup

[TPS_XMLSPR_00002] XML Schema production rule: xsd:attributeGroup ⌈

Applies to	Class without stereotype ≪enumeration≫
Precondition	Exists properties with "xml.attribute=true"

Target pattern	`<xsd:attributeGroup name="<%=xmlName%>">`
	`<% for all properties represented as XML attributes {`
	`<% call rule TPS_XMLSPR_00019 %>`
	`} %>`
	`</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.

XML schema example:	`<xsd:attributeGroup name="IDENTIFIABLE">`
	`<!--attributes defined by rule TPS_XMLSPR_00019 -->`
	`</xsd:attributeGroup>`

XML instance example:	n/a

⌋

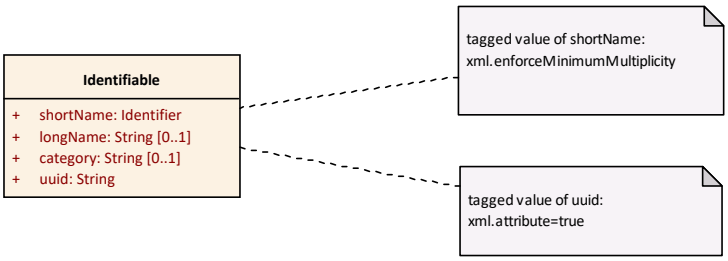

Figure 4.4: UML example - Create xsd:attributeGroup

4.2.3 Create xsd:complexType

[TPS_XMLSPR_00003] XML Schema production rule: xsd:complexType ⌈

Applies to	Class without stereotype ≪enumeration≫ or stereotype ≪primitive≫
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%>"/>`
	`  <% } %>`
	`  <% 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 super-classes (see rule [TPS_XMLSPR_00001] and [TPS_XMLSPR_00002]). The groups are ordered as defined in [TPS_XMLSPR_00033]. 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'.

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

⌋

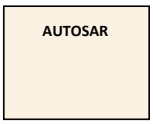

Figure 4.5: UML example - Create xsd:complexType

4.2.4 Create xsd:complexType with simple content

[TPS_XMLSPR_00024] XML Schema production rule: xsd:complexType with simple content ⌈

Applies to	Class without stereotype ≪enumeration≫ or stereotype ≪primitive≫
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. If the type of the property that is represented as attribute has properties, then these properties cannot be mapped to simpleContent. In this case an error shall be reported.

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`
	`</...>`

⌋

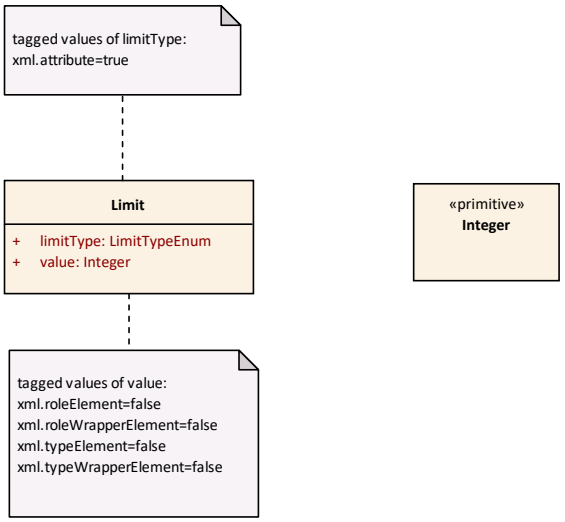

Figure 4.6: UML example - Create xsd:complexType with simple content

4.2.5 Create global xsd:element

[TPS_XMLSPR_00005] XML Schema production rule: global xsd:element ⌈

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 [TPS_XMLSPR_00003]. The namespace prefix is defined by the tagged value 'xml.nsPrefix'.
XML schema example:	`<xsd:element name="AUTOSAR" type="AR:AUTOSAR"/>`

XML instance example:	`<AUTOSAR>`
	`...`
	`</AUTOSAR>`

⌋

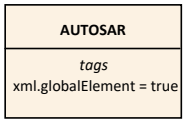

Figure 4.7: UML example - Create global xsd:element

4.2.6 Create enumeration of subtypes

[TPS_XMLSPR_00025] XML Schema production rule: enumeration of subtypes ⌈

Applies to	Class without stereotype ≪enumeration≫ or stereotype ≪primitive≫
Precondition	Class is referenced

Target pattern:	`<xsd:simpleType name="<%=xmlName%>--SUBTYPES-ENUM">`
	`  <xsd:restriction base="xsd:string">`
	`  <% for type and all subtypes { %>`
	`  <xsd:enumeration value="<%=typeXmlName%>"/>`
	`  <%} // for type and all subtypes %>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`

Description:	Creates an enumeration which represents the XML names of the class and all its subtypes. This enumeration is required for describing potential destination types of references.

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>`

⌋

4.2.7 Create reference to XML predefined data type

[TPS_XMLSPR_00006] XML Schema production rule: reference to XML predefined data type ⌈

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', unless a custom xsd:simpleType is defined by the tagged value 'xml.xsd.customType'. In the latter case rule [TPS_XMLSPR_00026] is applied. 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").
XML schema example:	The predefined W3C XML schema data type string is used to represent the primitive class String.
XML instance example:	`<... type="xsd:string" ...>`

⌋

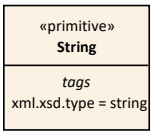

Figure 4.8: UML example - Create reference to XML predefined data type

4.2.8 Create a custom simple type

[TPS_XMLSPR_00026] XML Schema production rule: custom simple type ⌈

Applies to	class with stereotype ≪primitive≫
Precondition:	tagged value 'xml.xsd.customType=...' defined

Target pattern:	`<xsd:simpleType name="<%=xmlName%>--SIMPLE">`
	`  <xsd:restriction base="<%=xmlXsdType%>">`
	`  <xsd:pattern value="<%=xmlXsdPattern%>"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	`<xsd:complexType name="<%=xmlName%>">`
	`  <xsd:simpleContent>`
	`  <xsd:extension base="AR:<%=xmlName%>--SIMPLE">`
	`  <xsd:attributeGroup ref="AR:AR-OBJECT"/>`
	`  <% if (class has attributes) { %>`
	`  <xsd:attributeGroup ref="<%=xmlName%>" />`
	`  <% } // end if (class has attributes) %>`
	`  </xsd:extension>`
	`  </xsd:simpleContent>`
	`</xsd:complexType>`

Description:	Each class with stereotype ≪primitive≫ and tagged value xml.xsd.customType set is represented by two elements in the schema, a custom xsd:simpleType and a xsd:complexType with simple content. The name of the xsd:simpleType maps to the XML-name of the class suffixed by "--SIMPLE", the name of the xsd:complexType maps to XML-name of the class. The base for the simple type is the type defined in the tagged value xml.xsd.type. The restriction pattern is taken from the tagged value xml.xsd.pattern.

XML schema example:	`<xsd:simpleType name="POSITIVE-INTEGER--SIMPLE">`
	`  <xsd:restriction base="xsd:string">`
	`  <xsd:pattern value="[1-9][0-9]*|0x[0-9a-f]*|0[0-7]*|0b[0-1]*"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	`<xsd:complexType name="POSITIVE-INTEGER">`
	`  <xsd:simpleContent>`
	`  <xsd:extension base="AR:POSITIVE-INTEGER--SIMPLE">`
	`  <xsd:attributeGroup ref="AR:AR-OBJECT"/>`
	`  </xsd:extension>`
	`  </xsd:simpleContent>`
	`</xsd:complexType>`

XML instance example:	`<... type="AR:POSITIVE-INTEGER" ...>`

⌋

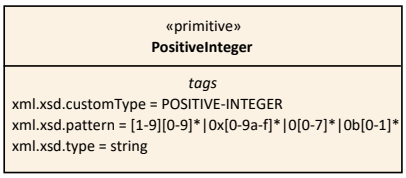

Figure 4.9: UML example - Create a custom simple type

4.2.9 Create xsd:simpleType for enumeration

[TPS_XMLSPR_00007] XML Schema production rule: xsd:simpleType for enumeration ⌈

Applies to:	class with stereotype ≪enumeration≫
Precondition:	n/a

Target pattern:	`<xsd:simpleType name="<%=xmlName%>--SIMPLE">`
	`  <xsd:restriction base="xsd:string">`
	`  <% for all attributes { %>`
	`  <xsd:enumeration value="<%=attributeXmlName%>"/>`
	`  <%} // end for all attributes %>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	`<xsd:complexType name="<%=xmlName%>">`
	`  <xsd:simpleContent>`
	`  <xsd:extension base="AR:<%=xmlName%>--SIMPLE">`
	`  <xsd:attributeGroup ref="AR:AR-OBJECT"/>`
	`  </xsd:extension>`
	`  </xsd:simpleContent>`
	`</xsd:complexType>`

Description:	Each class with stereotype ≪enumeration≫ is represented by two elements in the schema, a custom xsd:simpleType and an xsd:complexType with simple content. The name of the xsd:simpleType maps to the XML name of the class suffixed by "--SIMPLE", the name of the xsd:complexType maps to XML name of the class. The xsd:simpleType defines an enumeration. The enumeration literals are defined by the property names of the class.

XML schema example:	`<xsd:simpleType name="ENUMERATION-INFO-TYPE--SIMPLE">`
	`  <xsd:restriction base="xsd:string">`
	`  <xsd:enumeration value="DATA"/>`
	`  <xsd:enumeration value="EVENT"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	`<xsd:complexType name="ENUMERATION-INFO-TYPE">`
	`  <xsd:simpleContent>`
	`  <xsd:extension base="AR:ENUMERATION-INFO-TYPE--SIMPLE">`
	`  <xsd:attributeGroup ref="AR:AR-OBJECT"/>`
	`  </xsd:extension>`
	`  </xsd:simpleContent>`
	`</xsd:complexType>`

XML instance example:	`"data"`

⌋

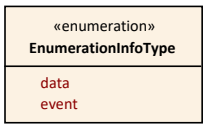

Figure 4.10: UML example - Create xsd:simpleType for enumeration

4.3 Create composite property representation (mapping to XML attributes)

As of [TPS_XMLSPR_00036] the rules in this section are only applicable to attributes as they are the only properties of a class that may be tagged with xml.attribute=true.

4.3.1 Create xsd:attribute

[TPS_XMLSPR_00019] XML Schema production rule: 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.
XML schema example:	`<xsd:attribute name="UUID" type="xsd:string" use="optional"/>`
XML instance example:	`<... UUID="12343-23342-12345-2333"/>`

⌋

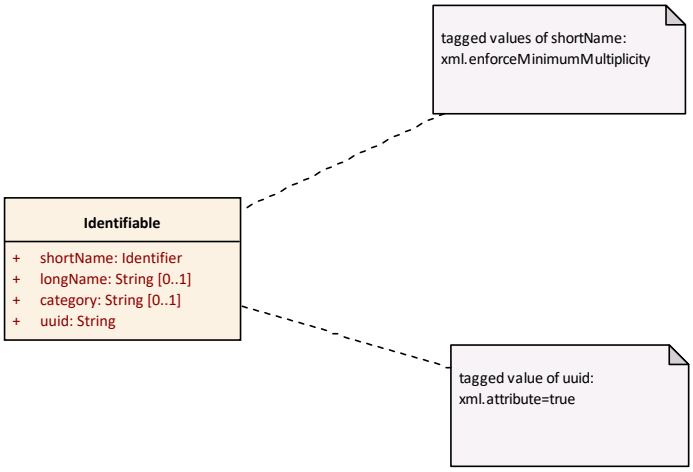

Figure 4.11: UML example - Create xsd:attribute

4.4 Create composite property representation (mapping to XML elements)

Composite properties are properties with 'aggregation=composite' (attributes and composition target ends). 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 4.12 illustrates how the rules are chosen based on the tagged values.

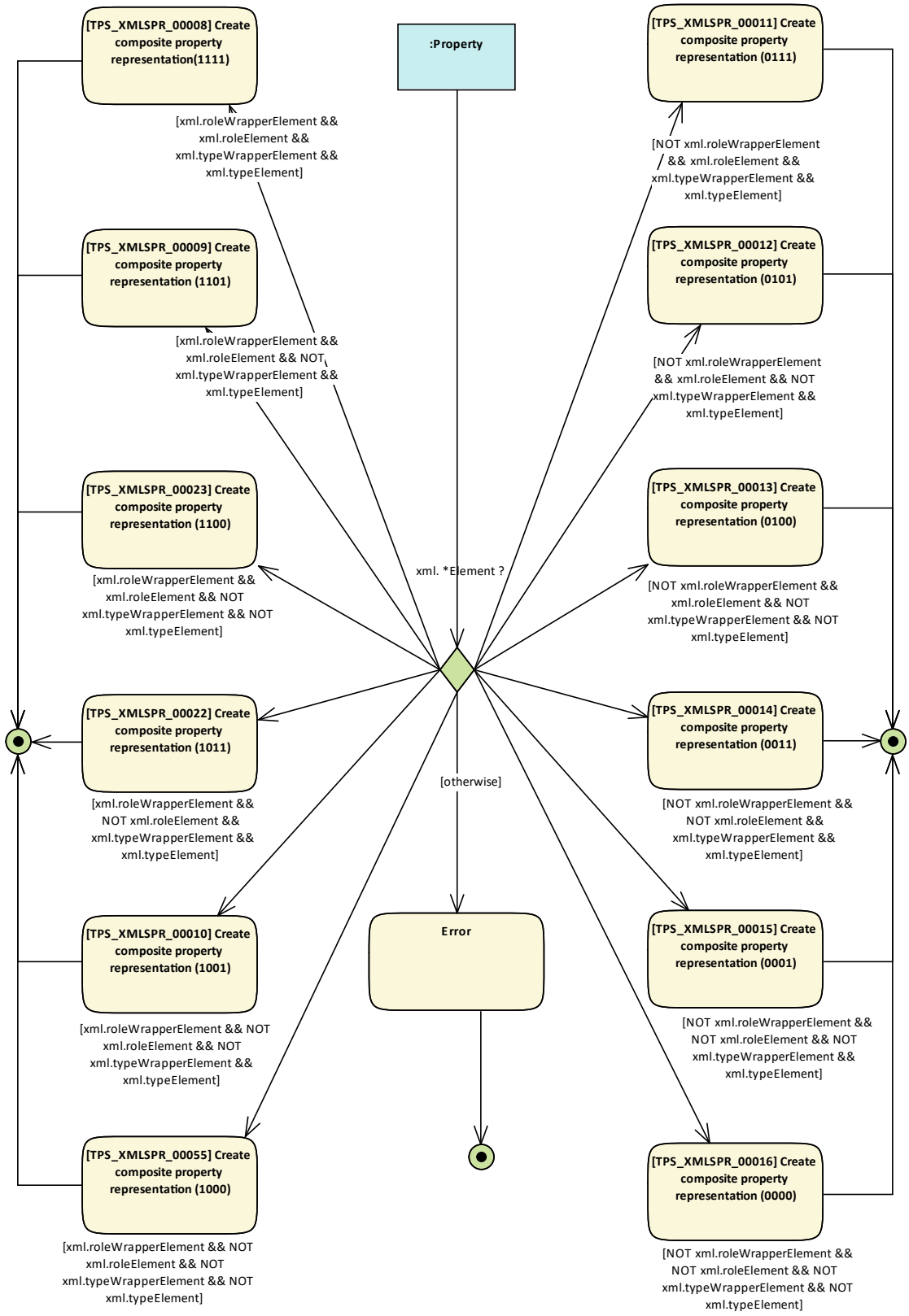

Figure 4.12: Property representation (aggregation = composite)

Figure 4.13 shows an example meta-model 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.

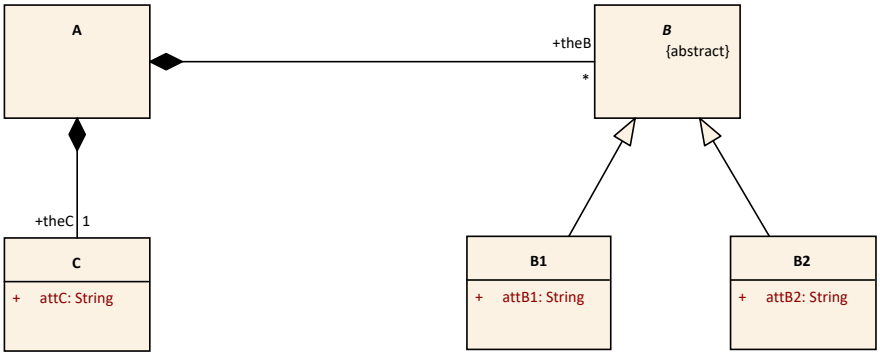

Figure 4.13: Example meta-model of composite property

Figure 4.14 shows an example instance of the meta-model shown in Figure 4.13. This instance is used as a basis for the XML instance examples of the following rules.

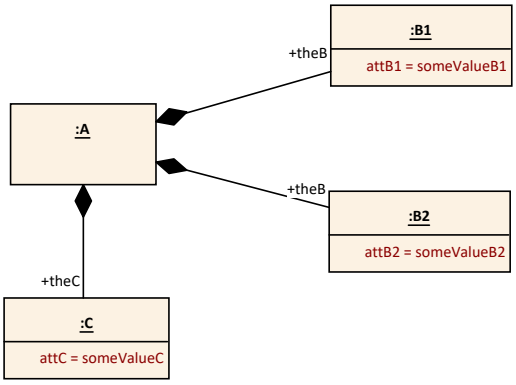

Figure 4.14: Example instance of composite property

4.4.1 Create composite property representation (1111)

[TPS_XMLSPR_00008] XML Schema production rule: composite property representation (1111) ⌈

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.

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>`
	`</...>`

⌋

4.4.2 Create composite property representation (1101)

[TPS_XMLSPR_00009] XML Schema production rule: composite property representation (1101) ⌈

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="0" 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.

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>`
	`</...>`

⌋

4.4.3 Create composite property representation (1100)

[TPS_XMLSPR_00023] XML Schema production rule: composite property representation (1100) ⌈

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.

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>
</THE-CS>
</...>
```

⌋

4.4.4 Create composite property representation (1011)

[TPS_XMLSPR_00022] XML Schema production rule: composite property representation (1011) ⌈

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.
XML schema example:	
XML instance 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>
```

```
<...>
  <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>
</.../>
```

⌋

4.4.5 Create composite property representation (1000)

[TPS_XMLSPR_00055] XML Schema production rule: composite property representation (1000) ⌈

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

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

Description:	An XML element is created that represents 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. The content model of the type directly shows up within the declaration of this element.

XML schema example:	`<xsd:element name="THE-BS" minOccurs="0" maxOccurs="1">`
	`  <xsd:complexType>`
	`  <xsd:choice minOccurs="0" maxOccurs="unbounded">`
	`  <xsd:group ref="AR:B1"/>`
	`  <xsd:group ref="AR:B2"/>`
	`  </xsd:choice>`
	`  </xsd:complexType>`
	`</xsd:element>`
	`<xsd:element name="THE-CS" minOccurs="0" maxOccurs="1">`
	`  <xsd:complexType>`
	`  <xsd:choice minOccurs="0" maxOccurs="unbounded">`
	`  <xsd:group ref="AR:C"/>`
	`  </xsd:choice>`
	`  </xsd:complexType>`
	`</xsd:element>`

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

⌋

4.4.6 Create composite property representation (1001)

[TPS_XMLSPR_00010] XML Schema production rule: composite property representation (1001) ⌈

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

Target pattern:	`<xsd:element name="<%=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.

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>
</...>
```

⌋

4.4.7 Create composite property representation (0111)

[TPS_XMLSPR_00011] XML Schema production rule: composite property representation (0111) ⌈

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.
XML schema example:	
XML instance 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>
```

```
<...>
  <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>`
	`</...>`

⌋

4.4.8 Create composite property representation (0101)

[TPS_XMLSPR_00012] XML Schema production rule: composite property representation (0101) ⌈

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

Target pattern:	`<xsd:element name="<%=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.

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>
</...>
```

⌋

4.4.9 Create composite property representation (0100)

[TPS_XMLSPR_00013] XML Schema production rule: composite property representation (0100) ⌈

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="0" 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.

▽

△

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>`
	`</...>`

⌋

4.4.10 Create composite property representation (0011)

[TPS_XMLSPR_00014] XML Schema production rule: composite property representation (0011) ⌈

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.

▽

△

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> </...>

⌋

4.4.11 Create composite property representation (0001)

[TPS_XMLSPR_00015] XML Schema production rule: composite property representation (0001) ⌈

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.

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>`
	`</...>`

⌋

4.4.12 Create composite property representation (0000)

[TPS_XMLSPR_00016] XML Schema production rule: composite property representation (0000) ⌈

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.

▽

△

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>`
	`</...>`

⌋

4.5 Create reference representation

Figure 4.15 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.

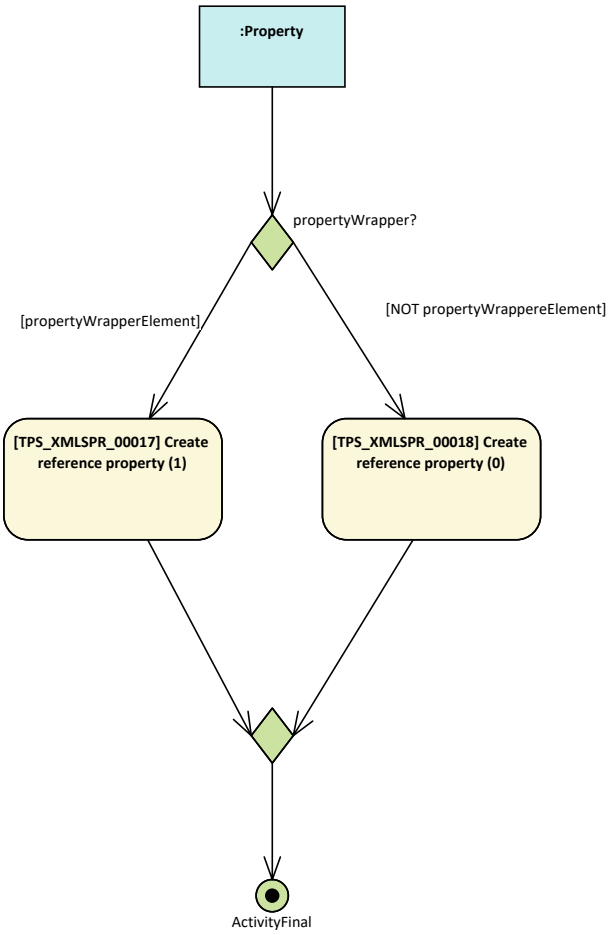

Figure 4.15: Property representation (aggregation = none)

Figure 4.16 shows an example meta-model that uses a reference. This example is used to illustrate the following two mapping rules.

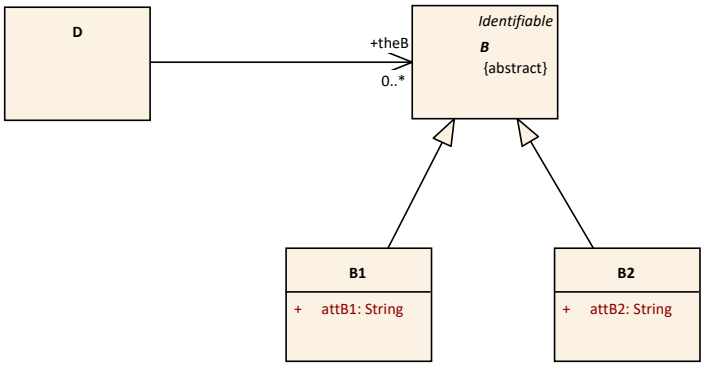

Figure 4.16: Example meta-model using a reference

Figure 4.17 shows an example instance of a reference. This example is used to illustrate the following two mapping rules.

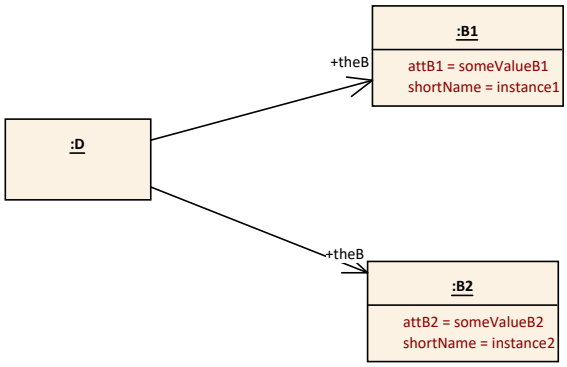

Figure 4.17: Example instance of reference

4.5.1 Create reference property representation (1)

[TPS_XMLSPR_00017] XML Schema production rule: reference property representation with role wrapper element ⌈

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): • An XML element that wraps several XML elements of the same XML name. The name of the wrapper is defined by the xml.namePlural of the property. Please note the default xml.namePlural is defined by the default singular XML name appended by "-REFS", "-TREFS" (see [TPS_XMLSPR_00043] and [TPS_XMLSPR_00045]). • An XML element that represents the reference itself. The name of this element is defined by the xml.name of the property. Please note the default xml.name is defined by the default singular XML name appended by by "-REF", "-TREF" (see [TPS_XMLSPR_00043] and [TPS_XMLSPR_00045]).

▽

△

XML schema example	`<xsd:element name="THE-B-REFS" > <xsd:complexType> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:element name="THE-B-REF"> <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> </...>`

⌋

4.5.2 Create reference property representation (0)

[TPS_XMLSPR_00018] XML Schema production rule: reference property representation without role wrapper element ⌈

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 note the default xml.name is defined by the default singular XML name appended by "-REF", "-TREF" (see [TPS_XMLSPR_00043] and [TPS_XMLSPR_00045]).

▽

△

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>`
	`</...>`

⌋

Remark: If the transformations specified in [TPS_GST_00351] are applied, this pattern will not occur.

4.5.3 Create a reference to attributes in foreign namespaces

[TPS_XMLSPR_00027] XML Schema production rule: reference to attributes in foreign namespaces ⌈

Applies to	Property
Precondition	xml.attributeRef=true
Target pattern	`<xsd:attribute ref="<%=nsPrefix%>:<%=xmlName%>"/>`
Description	An XML attribute that references attributes from foreign namespaces, e.g. the XML namespace. The name of the reference is defined by the xml.name of the property concatenated after the namespace prefix. Currently, only the XML namespace (nsPrefix=xml) is allowed.

XML schema example	`<xsd:attributeGroup name="WHITESPACE-CONTROLLED">`
	`  <xsd:attribute ref="xml:space" use="required">`
	`  <xsd:annotation>`
	`  <xsd:documentation>This attrbute is used to signal an intention`
	`  that in that element, white space should be preserved by`
	`  applications.`
	`  It is defined according to xml:space as declared by W3C.`
	`  </xsd:documentation>`
	`  </xsd:annotation>`
	`  </xsd:attribute>`
	`</xsd:attributeGroup>`

▽

△

XML instance example	

```
<ADMIN-DATA>
  <LANGUAGE>EN</LANGUAGE>
  <USED-LANGUAGES>
    <L-10 L="EN" xml:space="default">English</L-10>
  </USED-LANGUAGES>
</ADMIN-DATA>
```

⌋

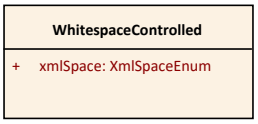

Figure 4.18: UML example - Create a reference to attributes in foreign namespaces

5 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.

A Change history of AUTOSAR traceable items

The content of this appendix chapter is *informative* in nature and shall **not** be considered as *normative* content.

A.1 Specification Item Mapping: Document migration from TR to TPS

In the course of the migration of this document from a Technical Report (TR) to a Template Specification (TPS) the specification item IDs were changed from TR_APRXML_XXXXX to TPS_XMLSPR_XXXXX. The exact mapping of specification item IDs is shown in table below.

Previous ID	Effective ID
TR_APRXML_00000	TPS_XMLSPR_00000
TR_APRXML_00001	TPS_XMLSPR_00001
TR_APRXML_00002	TPS_XMLSPR_00002
TR_APRXML_00003	TPS_XMLSPR_00003
TR_APRXML_00005	TPS_XMLSPR_00005
TR_APRXML_00006	TPS_XMLSPR_00006
TR_APRXML_00007	TPS_XMLSPR_00007
TR_APRXML_00008	TPS_XMLSPR_00008
TR_APRXML_00009	TPS_XMLSPR_00009
TR_APRXML_00010	TPS_XMLSPR_00010
TR_APRXML_00011	TPS_XMLSPR_00011
TR_APRXML_00012	TPS_XMLSPR_00012
TR_APRXML_00013	TPS_XMLSPR_00013
TR_APRXML_00014	TPS_XMLSPR_00014
TR_APRXML_00015	TPS_XMLSPR_00015
TR_APRXML_00016	TPS_XMLSPR_00016
TR_APRXML_00017	TPS_XMLSPR_00017
TR_APRXML_00018	TPS_XMLSPR_00018
TR_APRXML_00019	TPS_XMLSPR_00019
TR_APRXML_00022	TPS_XMLSPR_00022
TR_APRXML_00023	TPS_XMLSPR_00023
TR_APRXML_00024	TPS_XMLSPR_00024
TR_APRXML_00025	TPS_XMLSPR_00025
TR_APRXML_00026	TPS_XMLSPR_00026
TR_APRXML_00027	TPS_XMLSPR_00027
TR_APRXML_00028	TPS_XMLSPR_00028



△

Previous ID	Effective ID
TR_APRXML_00029	TPS_XMLSPR_00029
TR_APRXML_00030	TPS_XMLSPR_00030
TR_APRXML_00031	TPS_XMLSPR_00031
TR_APRXML_00032	TPS_XMLSPR_00032
TR_APRXML_00033	TPS_XMLSPR_00033
TR_APRXML_00034	TPS_XMLSPR_00034
TR_APRXML_00035	TPS_XMLSPR_00035
TR_APRXML_00036	TPS_XMLSPR_00036
TR_APRXML_00037	TPS_XMLSPR_00037
TR_APRXML_00038	TPS_XMLSPR_00038
TR_APRXML_00039	TPS_XMLSPR_00039
TR_APRXML_00040	TPS_XMLSPR_00040
TR_APRXML_00041	TPS_XMLSPR_00041
TR_APRXML_00042	TPS_XMLSPR_00042
TR_APRXML_00043	TPS_XMLSPR_00043
TR_APRXML_00044	TPS_XMLSPR_00044
TR_APRXML_00045	TPS_XMLSPR_00045
TR_APRXML_00046	TPS_XMLSPR_00046
TR_APRXML_00047	TPS_XMLSPR_00047
TR_APRXML_00054	TPS_XMLSPR_00054

Table A.1: Specification Item Mapping: Document migration from TR to TPS

A.2 Traceable item history of this document according to AUTOSAR Release R26-11

A.2.1 Added Specification Items in R26-11

none

A.2.2 Changed Specification Items in R26-11

none

A.2.3 Deleted Specification Items in R26-11

none

A.3 Traceable item history of this document according to AUTOSAR Release R25-11

A.3.1 Added Specification Items in R25-11

none

A.3.2 Changed Specification Items in R25-11

Number	Heading
[TPS_XMLSPR_00044]	XML Configuration of instance references
[TPS_XMLSPR_00056]	Default settings for the XML Configuration of properties

Table A.2: Changed Specification Items in R25-11

A.3.3 Deleted Specification Items in R25-11

none

A.4 Traceable item history of this document according to AUTOSAR Release R24-11

A.4.1 Added Specification Items in R24-11

none

A.4.2 Changed Specification Items in R24-11

none

A.4.3 Deleted Specification Items in R24-11

none

A.5 Traceable item history of this document according to AUTOSAR Release R23-11

A.5.1 Added Specification Items in R23-11

Number	Heading
[TPS_XMLSPR_00057]	Constraints on tags applied to properties
[TPS_XMLSPR_00058]	Constraints on tags applied to classes
[TPS_XMLSPR_00059]	Constraints on values of xml.*Element tagged values

Table A.3: Added Specification Items in R23-11

A.5.2 Changed Specification Items in R23-11

none

A.5.3 Deleted Specification Items in R23-11

none

A.6 Traceable item history of this document according to AUTOSAR Release R22-11

A.6.1 Added Specification Items in R22-11

none

A.6.2 Changed Specification Items in R22-11

Number	Heading
[TPS_XMLSPR_00001]	XML Schema production rule: xsd:group
[TPS_XMLSPR_00002]	XML Schema production rule: xsd:attributeGroup
[TPS_XMLSPR_00003]	XML Schema production rule: xsd:complexType
[TPS_XMLSPR_00024]	XML Schema production rule: xsd:complexType with simple content
[TPS_XMLSPR_00025]	XML Schema production rule: enumeration of subtypes

Table A.4: Changed Specification Items in R22-11

A.6.3 Deleted Specification Items in R22-11

none

A.7 Traceable item history of this document according to AUTOSAR Release R21-11

A.7.1 Added Specification Items in R21-11

none

A.7.2 Changed Specification Items in R21-11

Number	Heading
[TPS_XMLSPR_00026]	XML Schema production rule: custom simple type
[TPS_XMLSPR_00028]	Optional XML elements
[TPS_XMLSPR_00044]	XML Configuration of instance references

Table A.5: Changed Specification Items in R21-11

A.7.3 Deleted Specification Items in R21-11

none