Document Title	Generic Structure Template
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	202
Document Classification	Standard

Document Version	3.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History			
Date	**Version**	**Changed by**	**Description**
31.10.2011	3.2.0	AUTOSAR Administration	• Editorial changes including tagged specification items • Improvements in UML usage (M3), especially mark obsolete elements • Improved specification of primitives, primitive definition, formula language, category • Improved variant handling and blueprint support • Improved support for instanceRef and arrays • Improved definition of package structures

14.10.2010	3.1.0	AUTOSAR Administration	• Editorial changes • Improvements in variant handling (Package content, composed predefined variants) • Align Formula language with ASAM General Expression Language • Generalized approach for anntoations • Improved aligment with ASAM - FSX • Document the admin.* uml tags. • Support global referenceing and tracing
30.11.2009	3.0.0	AUTOSAR Administration	• restructured the document • support for variant handling • support for abstract structures • documentation support • detailed primitives • general modeling information required to understand other templates
23.06.2008	2.1.1	AUTOSAR Administration	• Legal disclaimer revised • Rename document from "Template Modeling Patterns" to "Generic Structure Template"
20.12.2007	2.1.0	AUTOSAR Administration	• Updated Attributes of Identifiable • Added "Hint to the Users" • Added document identification no • Added document classification
07.02.2007	2.0.2	AUTOSAR Administration	• Legal disclaimer revised • "Advice for users" revised • "Revision Information" added
27.06.2006	2.0.1	AUTOSAR Administration	Layout Adaptations

19.05.2006	2.0.0	AUTOSAR Administration	Second release
18.10.2005	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

Advice for users

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

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

Table of Contents

References

[1] Meta Model
AUTOSAR_MMOD_MetaModel.eap

[2] Model Persistence Rules for XML
AUTOSAR_TR_XMLPersistenceRules.pdf

[3] Specification of ECU Resource Template
AUTOSAR_TPS_ECUResourceTemplate.pdf

[4] System Template
AUTOSAR_TPS_SystemTemplate.pdf

[5] AUTOSAR Template Modeling Patterns
AUTOSAR_TemplateModelingPatterns.pdf

[6] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[7] Predefined Names
AUTOSAR_TR_PredefinedNames.pdf

[8] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[9] Standardization Template
AUTOSAR_TPS_StandardizationTemplate.pdf

[10] ANTLR parser generator V3

[11] C++ Operator Precedence
http://www.cppreference.com/wiki/operator_precedence

[12] Issue Exchange Format V3.0.0
http://www.asam.net

[13] Container Catalog XML Model Specification
http://www.asam.net

[14] Specification of RTE Software
AUTOSAR_SWS_RTE.pdf

[15] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate.pdf

[16] ASAM AE Functional Specification Exchange Format V1.0.0
http://www.asam.net
AE-FSX_V1.0.0.pdf

[17] OASIS open exchange table model
http://www.oasis-open.org/specs/tm9901.html

[18] Software Process Engineering Meta-Model Specification
http://www.omg.org/spec/SPEM/2.0/

1 Introduction

This document contains the specification of the AUTOSAR `Generic Structure Template`. Actually, it has been created as a supplement to the formal definition provided by the AUTOSAR meta-model [1]. In other words, this document in addition to the formal specification provides introductory description and rationale for the parts of the AUTOSAR meta-model relevant for almost all AUTOSAR templates.

Nevertheless, the core part of the specification is directly based on the content of the AUTOSAR meta-model. Therefore, this document contains a summary of the main concepts of the AUTOSAR meta-model, see chapters 1.3 and 1.4.

This document provides reference information and is not intended to be read in a sequence. Nevertheless it contains as major aspects:

1. Chapter 3 explains the top level structure which is common to all AUTOSAR templates.
2. Mechanisms used to design AUTOSAR templates:
   (a) Chapter 2 describes an essential aspects of the Autosar Template UML profile which are necessary to understand the AUTOSAR template documents.
   (b) Chapter 4 describes general template classes which are collected similar to the standard library of a compiler.
   (c) Chapter 5 explains abstract classes with abstract relationships. These structures implement **particular concepts** applicable to all Autosar templates. These concepts are applied by specializing these abstract classes and in particular specializing the abstract relationships.
   (d) Chapter 6 explains in general the approach to apply by model transformation (as for example used for Variant handling).
3. Some specific applications of design mechanisms in the MetaModel
   (a) Chapter 7 describes the implementation of variant handling within Autosar templates based on MetaModeling Patterns (as described in 6).
   (b) Chapter 8 describes the documentation support.

1.1 Scope

The scope of this document covers information which is required to understand the AUTOSAR templates and the core mechanisms used to define these templates.

Aspects of UML modeling required to perform the template Modeling tasks are out of the scope of this document.

1.2 Document Conventions

Technical terms are typeset in mono spaced font, e.g. `PortPrototype`. As a general rule, plural forms of technical terms are created by adding "s" to the singular form, e.g. `PortPrototype`s. By this means the document resembles terminology used in the AUTOSAR XML Schema.

This document contains constraints in textual form that are distinguished from the rest of the text by a unique numerical constraint ID, a headline, and the actual constraint text starting after the ⌈ character and terminated by the ⌋ character.

The purpose of these constraints is to literally constrain the interpretation of the AUTOSAR meta-model such that it is possible to detect violations of the standardized behavior implemented in an instance of the meta-model (i.e. on M1 level).

Makers of AUTOSAR tools are encouraged to add the numerical ID of a constraint that corresponds to an M1 modeling issue as part of the diagnostic message issued by the tool.

The attributes of the classes introduced in this document are listed in form of class tables. They have the form shown in the example of the top-level element AUTOSAR:

Class	**AUTOSAR**			
Package	M2::AUTOSARTemplates::AutosarTopLevelStructure			
Note	Root element of an AUTOSAR description, also the root element in corresponding XML documents. **Tags:** xml.globalElement=true			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data of an Autosar file. **Tags:** xml.sequenceOffset=10
arPackage	ARPackage	*	aggr	This is the top level package in an AUTOSAR model. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=BlueprintDerivation Time atp.Splitkey=shortName xml.sequenceOffset=30
introductio n	Documentation Block	0..1	aggr	This represents an introduction on the Autosar file. It is intended for example to rpresent disclaimers and legal notes. **Tags:** xml.sequenceOffset=20

Table 1.1: AUTOSAR

The first rows in the table have the following meaning:

Class: The name of the class as defined in the UML model.

Package: The UML package the class is defined in. This is only listed to help locating the class in the overall meta model.

Note: The comment the modeler gave for the class (class note). Stereotypes and UML tags of the class are also denoted here.

Base Classes: If applicable, the list of direct base classes.

The headers in the table have the following meaning:

Attribute: The name of an attribute of the class. Note that AUTOSAR does not distinguish between class attributes and owned association ends.

Datatype: The datatype of an attribute of the class.

Mul.: The assigned multiplicity of the attribute, i.e. how many instances of the given data type are associated with the attribute.

Kind: Specifies, whether the attributes is aggregated in the class (`aggr`), an UML attribute in the class (`attr`), or just referenced by it (`ref`). Instance references are also indicated (`iref`) in this field.

Note: The comment the modeler gave for the class attribute (role note). Stereotypes and UML tags of the class are also denoted here.

1.3 Methodology for Defining Formal Templates

Figure 1.1 illustrates the overall methodology used to define formal templates using System Template as an example. A precise and concise model of the information that needs to be captured in AUTOSAR XML files is provided in [1]

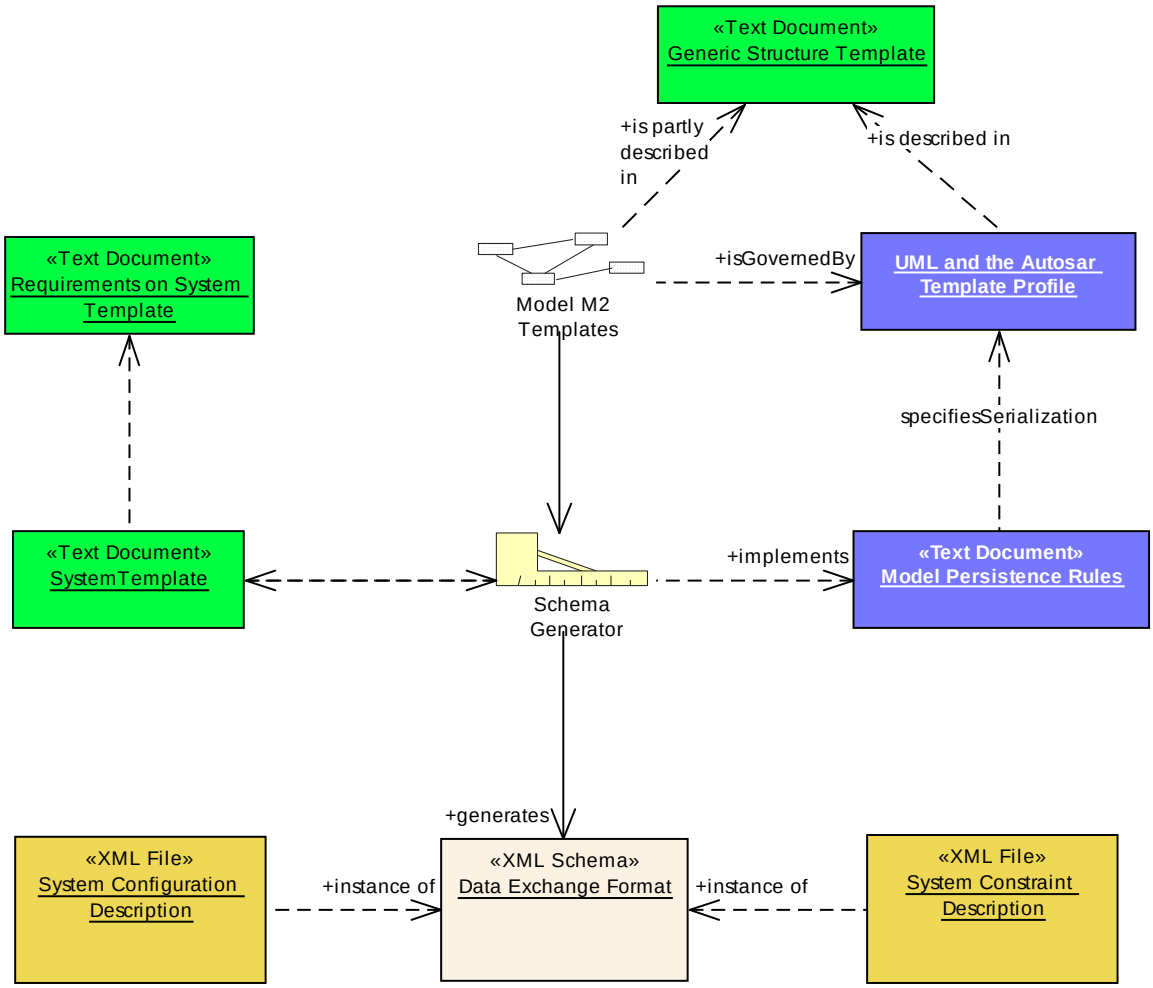

Figure 1.1: Methodology to define templates in AUTOSAR (using SystemTemplate as example

The following documents describe the various aspects of the methodology:

1. The **template document** (in this example `System Template`) describes the information that can be captured in the template, independently from the mapping of this model on XML-technology. It contains an elaborate description of the semantics (the precise meaning) of all the information that can be captured within the relevant parts of the AUTOSAR meta-model.
2. The model called **M2 Templates** in the AUTOSAR meta-model [1] contains the structure of the AUTOSAR templates modeled in UML. The model is annotated using notes which are also represented as class tables in the template documents.
3. The document called **Generic Structure Template** (this document) is represented e.g. as predefined Classes in the meta-model which are incorporated in the generated schema.

4. The **Template UML Profile and Modeling Guide** describes the basic concepts that were applied when creating content of the meta-model. This information is presented in chapter 2.
5. The document called **Model Persistence Rules for XML** [2] describes how XML is used and how the meta-model designed in the "Software Component Template" should be translated by the "Schema Generator" (MDS) into XML-Schema (XSD) "Data Exchange Format".
   This "formalization strategy" is supposed to be used for all data that is formally described in the meta-model. In particular this document is worth to read in order to understand the mapping of the meta-model and the XML based AUTOSAR template.
6. The **Data Exchange Format** is represented as an XML schema automatically generated out of the AUTOSAR meta-model using the approach and the patterns defined in the **Model Persistence Rules for XML**. This schema is typically used as input to AUTOSAR tools.
7. The **M1-level descriptions** (in figure 1.1 illustrated as "System configuration description" and "System Constraint Description") are XML files that can be validated against the XML schema and furtheron follow the specifications in the relevant "template document". In other words, the XML files are instances of the schema defining the XML representation of the template.

1.4 Organization of the Meta-Model

Figure 1.2 sketches the overall structure of the meta-model, which formally defines the vocabulary required to describe AUTOSAR software-components. As the diagram points out, other template specifications (e.g. `ECU Resource Template` [3] and `System Template` [4]) also use the same modeling approach in order to define an overall consistent model of AUTOSAR software description.

The dashed arrows in the diagram describe dependencies in terms of import-relationships between the packages within the meta-model. For example, the package `SWComponentTemplate` imports meta-classes defined in the packages `Generic-Structure` [5] and `ECUResourceTemplate` [3].

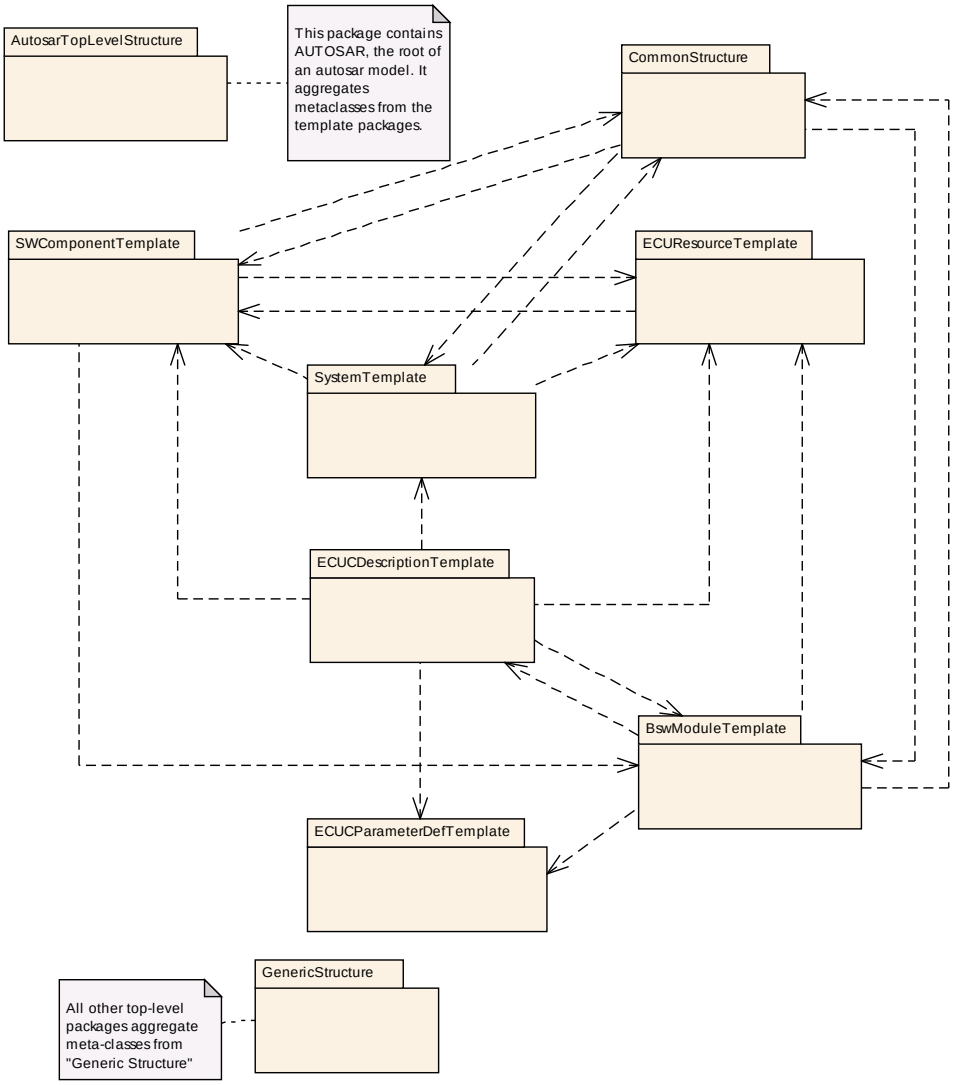

Figure 1.2: Structure of the meta-model

For clarification, please note that the package `GenericStructure` contains some fundamental infrastructure meta-classes and common patterns that are described in [5]. As these are used by all other template specification the dependency associations are not depicted in the diagram for the sake of clarity.

2 Usage of UML in AUTOSAR Templates

The AUTOSAR meta-model is defined as an UML model. Therefore basic knowledge of UML is required to understand the AUTOSAR template documents.

2.1 UML Diagrams

The diagrams in the AUTOSAR Template documents are consistent with UML 2.0. The underlying model (the AUTOSAR metamodel) is assumed to be complete even though certain elements might not be shown in a particular diagram to simplify understanding. Nevertheless the class tables show all relevant information.

The coloring of the diagrams is usually explained in the surrounding text. But in general the classes in light green color are those which are taken from MSR.

2.2 The AUTOSAR Meta-Model Hierarchy

The complete meta model hierarchy for AUTOSAR templates is shown in figure 2.1. Unlike the classical four-layer architecture used by OMG, five meta levels are shown. Starting at the lowest, most concrete meta level those are:

- **M0: AUTOSAR objects**
  This is the realization of an AUTOSAR system at work: For example a real ECUs executing a software image containing for instance the windshield wiper control software.
- **M1: AUTOSAR models**
  Models on this meta level are built by the AUTOSAR developers. They may define a software component called "windshield wiper" with a certain set of ports that is connected to another software component and so on. On this level all artifacts required to describe an AUTOSAR system are detailed, including re-usable types as well as specific instances of such types.

  The AUTOSAR software is loaded in to individual ECUs for individual vehicles. This loading means that the M1 Model is instantiated.

  Note that such an AUTOSAR model can be represented using various formats ranging from XML, to C even to PDF.
- **M2: AUTOSAR meta-model**
  On this meta level the vocabulary for AUTOSAR templates is defined. This vocabulary later can be used by developers of AUTOSAR based ECU systems.

  For example it is **defined on M2** that in AUTOSAR we have an entity called "software component" which among others aggregate an entity called "port". This definition ensures that the developer of an AUTOSAR software component can

describe his particular component and its ports. This description is called an AUTOSAR model and **resides on M1**.

- **M3: UML profile for AUTOSAR templates**
  The AUTOSAR templates on M2 are built according to the meta-model defined on M3. As discussed before this is UML together with a particular UML profile to better support template modeling work.

  Formally a template on M2 is still an instance of UML, but at the same time the template profile is applied, i.e. that additionally rules set out by the stereotypes in the profile need to be observed. The relevant details of the profile are specified in chapter 2.3 and chapter 2.4.

Note that an AUTOSAR model can be represented using various formats ranging from XML, to C even to PDF. The conversion between these formats is called "transformation", while the fact that an AUTOSAR model follows to the AUTOSAR meta-model is called "instantiation". An AUTOSAR model (M1) is therefore called an instance of **the** AUTOSAR meta-model (M2).

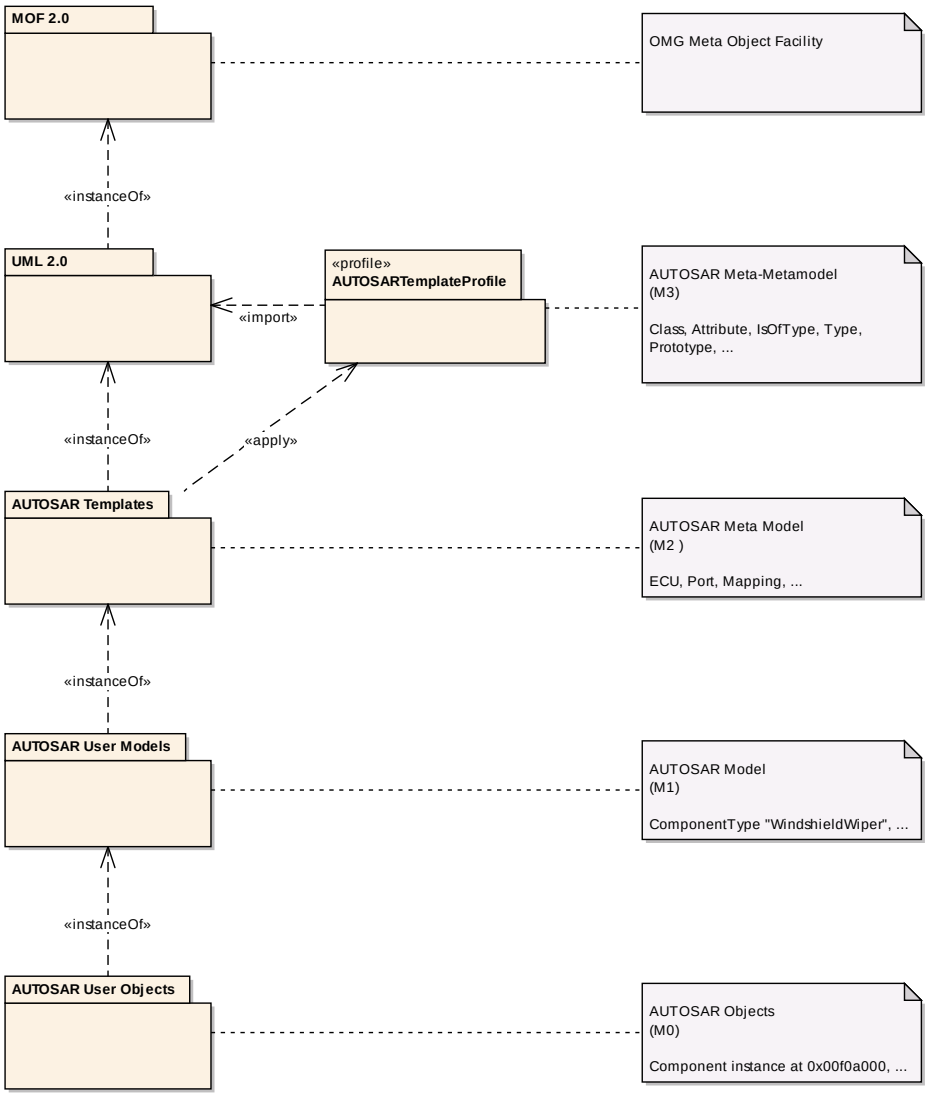

Figure 2.1: Meta Model Hierarchy

2.3 Stereotypes

The AUTOSAR Template Profile uses the following stereotypes[1]:

- **[TPS_GST_0022] ≪atpAbstract≫ applicable to relations (associations, aggregations)** ⌈ This indicates that the relationship is abstract. There needs to be specialized relation in every concrete subclass redefining the abstract relation. This stereotype is there to provide a better visualization in the diagrams. The fact that the relation is abstract is modeled by defining the role as "derived" in the model. It is also indicated by a "/" in front of the role name in the diagrams.

  Relations of ≪atpAbstract≫ exist only in the superclass and are not inherited to subclasses. They *need to be redefined* in the subclasses[2]. ⌋

- **[TPS_GST_0023] ≪atpDerived≫ applicable to relations (associations, aggregations)** ⌈ This indicates that the relationship exists in the subclasses by inheritance. It further indicates that in M1 models it is calculated (derived) from other information.

  There are two types of calculation:

  - **general**
    indicates that the value is calculated by a method which is described in the note of the abstract relation. For example the atpBase is calculated as the container of the first atpContextElement.
  - **derived union**
    indicates that it is derived as the union of all concrete relations.

    For example, the aggregation from AtpClassifier to AtpFeature with role atpFeature is ≪atpDerived≫, SwComponentType has an atpFeature association **in addition** to component, port etc. This atpFeature is calculated as the union of the concrete features.C

    This means that for a given component type, its atpFeature property holds its ports AND its contained component prototypes AND its contained connectors. This allows to define the instance reference on abstract level.

  Refer to chapter 5 for further details. ⌋

- **[TPS_GST_0024] ≪atpMixed≫ applicable to classes** ⌈

  This is applied to meta-classes only and indicates a mixed content model **without** intermixed text. ⌋ For more details see chapter 2.3.1.

- **[TPS_GST_0025] ≪atpMixedString≫ applicable to classes** ⌈

  This is a mixed content model **with** intermixed text. This is applied to meta-classes only. ⌋ For more details see chapter 2.3.1.

[1] the names of these stereotypes start with atp which is the abbreviation for **A**utosar **T**emplate **P**rofile
[2] In consequence of such a redefinition the XSD-generator ignores such abstract relations.

- **[TPS_GST_0026]** ≪**atpObject**≫ **applicable to classes** ⌈

  This is an implicit base class. It can only provide attributes with tagged with `xml.attribute=true`. ⌋ For more details see chapter 6.3.3.

- **[TPS_GST_0027]** ≪**atpSplitable**≫ **applicable to relations** ⌈ By using the stereotype ≪atpSplitable≫ the meta model can explicitly define how instances of the meta model may be distributed over several files. By default all data is stored in one single file. If the ≪atpSplitable≫ is applied, then the associated or aggregated information may be stored in different files. For more details see chapter 2.3.2. ⌋

- **[TPS_GST_0028]** ≪**atpVariation**≫ **applicable to classes and relations** ⌈ This indicates variant handling. It is applied to meta-classes as well as to associations or aggregations. ⌋

  For more details see chapter 7.

- **[TPS_GST_0029]** ≪**atpUriDef**≫ **applicable to associations** ⌈ This indicates that the essential information is only the full qualified name of the reference.target. This is then used as a whole as identifier for a particular purpose. The association acts as the definition of a kind of "Universal Resource Identifier" in the AUTOSAR model.

  Note that in this case only the full qualified shortName path is important, and not the `shortName` of the target nor the target itself. The particular semantics and therefore the subsequent processing depends on the individual use case. ⌋

  Therefore it is not always necessary to really follow the references of stereotype ≪atpUriDef≫. Tools should not warn about dangling references of this stereotype unless explicitly requested by the user respectively the particular use case.

  For example in `EcucReferenceDef.destination` the reference indicates that valid targets of the `EcucReferenceValue` must be `EcucContainerValues` whose definition is derived from the target of `EcucReferenceDef.destination`. But this can be verified even if the target `EcucReferenceDef.destination` is not really available.

- **[TPS_GST_0030]** ≪**instanceRef**≫ **applicable to dependencies** ⌈ This is used to provide a simplified representation of instance references within diagrams. See chapter 5.1.3 for more details. ⌋

- **[TPS_GST_0031]** ≪**isOfType**≫ **applicable to associations** ⌈ This is used to emphasize the concrete relationship between prototypes (subclasses of `AtpPrototype` and types (subclasses of `AtpType`). ⌋

  This stereotype influences in generation of associations according to chapter 6.3.2[3].

[3]Note that this stereotype is redundant to the fact that such associations need to redefine the role `atpType` directly or indirectly.

2.3.1 Mixed Content (≪atpMixed≫, ≪atpMixedString≫)

If a meta-class has several attributes (which may include aggregations or references), there are cases in which the “serialized” representation (like XML) in the M1 model adds semantics to the actual order and to the number of occurrences of the attribute instances. It may be also be required, that the same attribute appears in multiple instances (in M1) which are mixed with other attribute instances. This situation cannot be expressed in UML in a simple way.

In addition, if a model requires to describe documentation-like information, it often will need to mix formal content and text. An example of such a model is a an embedded link in HTML: markup of formal information bits is mixed into regular text, as shown in the following example:

```
[...]meet <a href="/wiki/Runtime" title="Runtime">runtime</a> requirements
of automotive devices[...]
```

This example illustrates that the "mixed content" feature is is well known in the XML world, where it is called mixed content[4].

[TPS_GST_0032] Basic Features of Mixed Content ⌈ The following list indicates the features of mixed content from a modeling point of view. Within a mixed content instance

- a set of formally defined model elements may appear an arbitrary number of times in arbitrary order
- but the actually present order is relevant in terms of semantics of the whole object, and
- In case of ≪atpMixedString≫ unqualified text may be mixed in between any of formally defined elements.

⌋

This mechanism is supported in AUTOSAR through stereotypes

- ≪atpMixedString≫ which allows text between the data elements
- ≪atpMixed≫ which allows any mix of the properties of such a class in any order

The latter stereotype does not allow for mixed-in text, but keeps the definition of order in terms of syntax and semantics.

[TPS_GST_0033] Upper Multiplicity in Mixed content ⌈ A mixed content class will aggregate or reference a number of other classes in a template model. The target multiplicity of those relations are typically 1, since the overall number of occurrences is arbitrary by definition given in [TPS_GST_0032].

If, however, multiplicity is different from 1, a required grouping is specified. ⌋

[4] http://www.w3schools.com/schema/schema_complex_mixed.asp

For example if the target multiplicity is 2, always a pair of those objects (not just a single object) must be put into the mixed content, and so on.

Figure 2.2 illustrates how it works. The M1 Model is shown in XML. Please note

1. `MixedContent` can be an arbitrary mix any order of `a b c d e` in any order. The order is semantically important. This is the same significance as if an aggregation of upper multiplicity > 1 is annotated as `{ordered}` in UML[5].
2. `c` is of upper multiplicity $>$ 1 therefore the wrapper for multiplicity is there
3. `e` is legally missing since overall number of occurrences is arbitrary by definition.

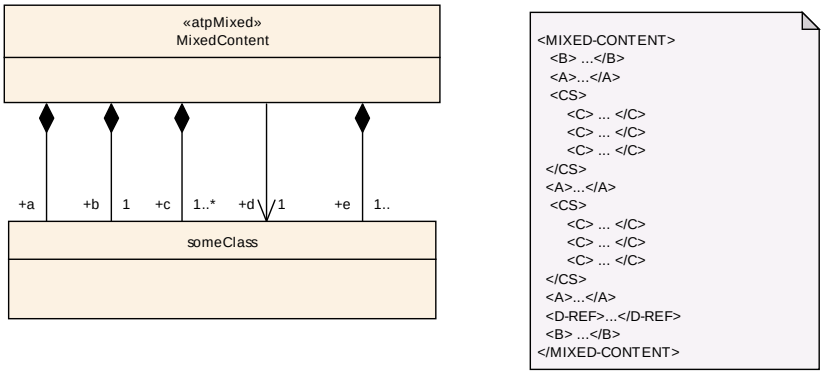

Figure 2.2: Mixed content

Please note that attributes and inherited properties are **not** part of the mixed content.

2.3.2 Splitable Elements (≪`atpSplitable`≫) distributed on Multiple Physical Files

AUTOSAR distinguishes between elements in the model that may be split up over several physical files, and elements that need to be defined together and therefore shall be described completely in exactly one physical file. By default all properties of an element must be in the same physical file. However for each individual property (attribute, aggregation and reference) the meta model can specify by the stereotype ≪`atpSplitable`≫ that it can be distributed in separate files. These files are merged into one “merged model”.

[constr_2502] Merged model must be compliant to the meta-model. ⌈ A model merged from ≪`atpSplitable`≫ elements must adhere to the consistency rules of the *pure meta model*. Especially, the multiplicities in the bound model must conform to the multiplicities and the constraints of the *pure meta-model*. ⌋

By introducing splitable elements, AUTOSAR supports

- Flexible Methodology (e.g. optimizing processes by specific distribution of Information to physical artifacts)

[5]UML it is not possible to denote this annotation for classes.

- to add further aspects to an element without changing the original artifact

The semantics of ≪atpSplitable≫ is:

- If the meta model marks an *aggregation*/*attribute* as ≪atpSplitable≫ then the aggregated elements may be described in different physical files (representing partial models).

  [constr_2524] Non splitable elements in one file ⌈ If the *aggregation*/*attribute* is **not** ≪atpSplitable≫, then all aggregated element(s) shall be described in the same physical file as the aggregating element. ⌋

  If a ≪atpSplitable≫ aggregation is of upper multiplicity > 1 we have a collection of elements. This collection then may be split to different files. The partial models represented by these different files still provide collections contributing to the merged model. Therefore according to [2], the wrapper xml element shall be given in each individual file as well.

  Note that ≪atpSplitable≫ on ordered collections (relations with upper multiplicity > 1) indicates that the entire collection could be in a partial model.

  [constr_2547] ordered collections cannot be split into partial models ⌈ Ordered collections which are splitable must be in one partial model as a whole. In other words: In opposite to unordered collections - which can be distributed between partial models - ordered collections can only be placed as a whole in one of the partial models. Otherwise the merge approach would influence the semantics of the collections. ⌋

- **[constr_2525] Non splitable elements shall not be repeated** ⌈ Properties (namely aggregations and attributes) which are **not** marked as ≪atpSplitable≫ must be all together in one physical file. They must not be repeated in the split files unless they are required for proper merging. ⌋

In order to achieve this, the following is held true in the meta model:

- the ≪atpSplitable≫ property can be unambiguously identified by one of the following properties (decreasing precedence). Such properties must be repeated in the split files.
  - shortName of aggregated element (if upper multiplicity of aggregation is > 1)
  - if the ≪atpSplitable≫ property is subject to variation (see chapter 7), the shortLabel of the VariationPoint needs to be considered as well.
  - Role of aggregation (only for relationships of upper multiplicity equal to 1)
  - additional means, for example definition within EcucParameterValue or within EcucContainerValue of an ECU parameter.

  The particular properties are specified explicitly as comma separated list in the UML tag atp.Splitkey.

- if an element contains properties marked as ≪atpSplitable≫ then all aggregations up to the root element are marked as ≪atpSplitable≫. This allows to unambiguously identify split elements in all involved physical files.

The following pages illustrates how a model can be distributed across multiple files. Figure 2.3 shows the meta model for the example (M2). The merged model is shown in figure 2.4. This diagram also indicates through colors how the elements are distributed across four files (indicated by different colors). Note that in figure 2.4 ≪atpSplitable≫ is shown for better understanding even if the diagram shows an M1 artifact (while stereotypes are applied on M2).

Two examples for invalid splitting is given in

- Figure 2.5: nonSplitDataSpec is missing.
- Figure 2.6: attribute a must be together with the attribute b in one file, because these attributes are not splitable. The attribute a is also not part of the split key.

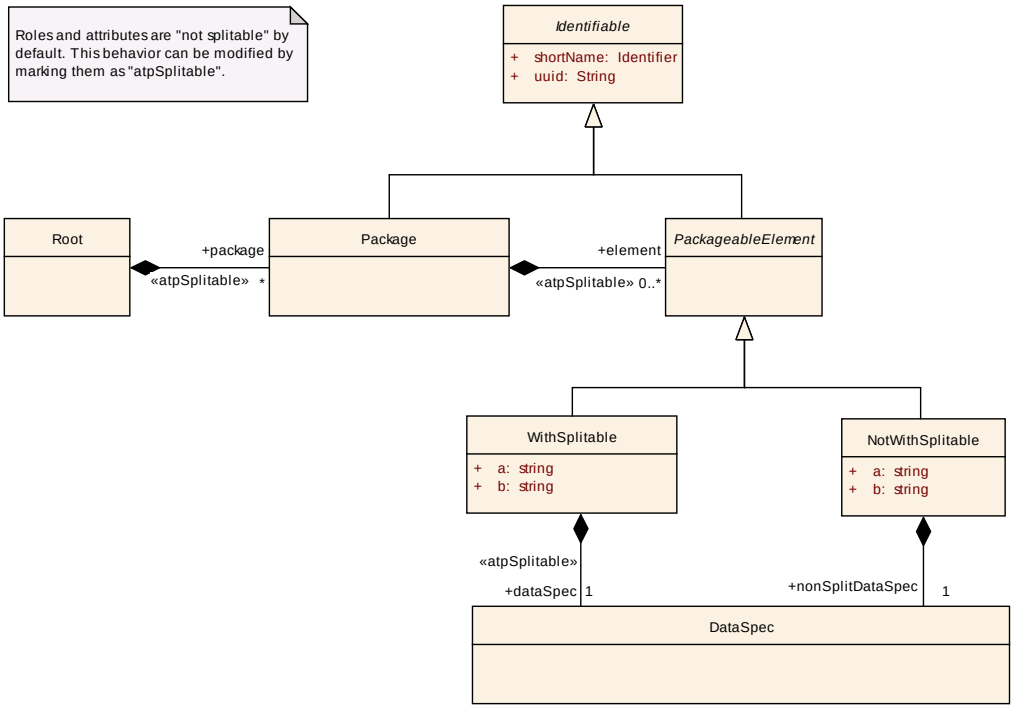

Figure 2.3: Metamodel with splitables

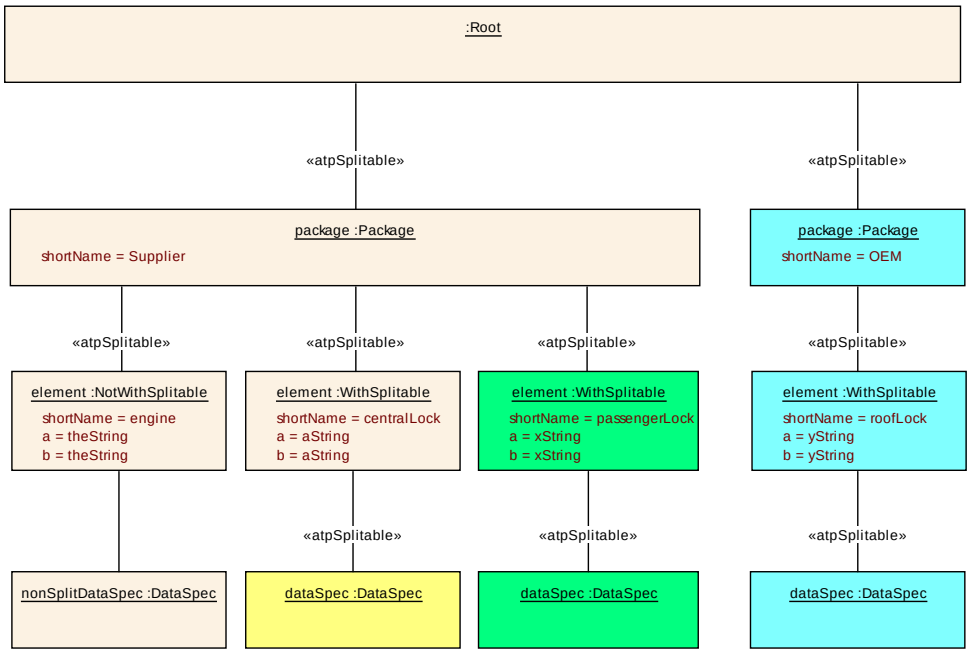

Figure 2.4: Final model after merging

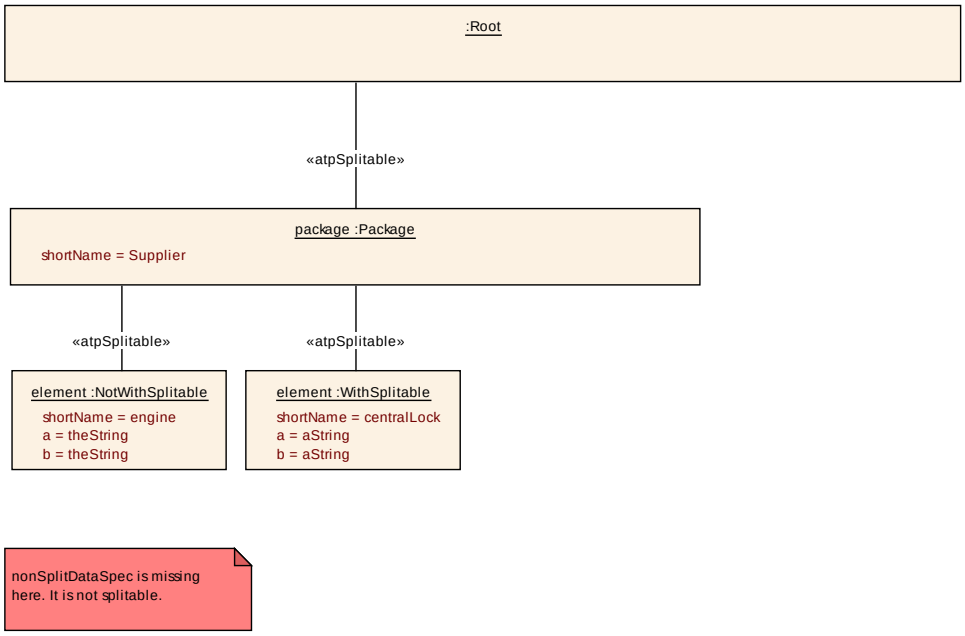

Figure 2.5: Invalid Partial Model (1)

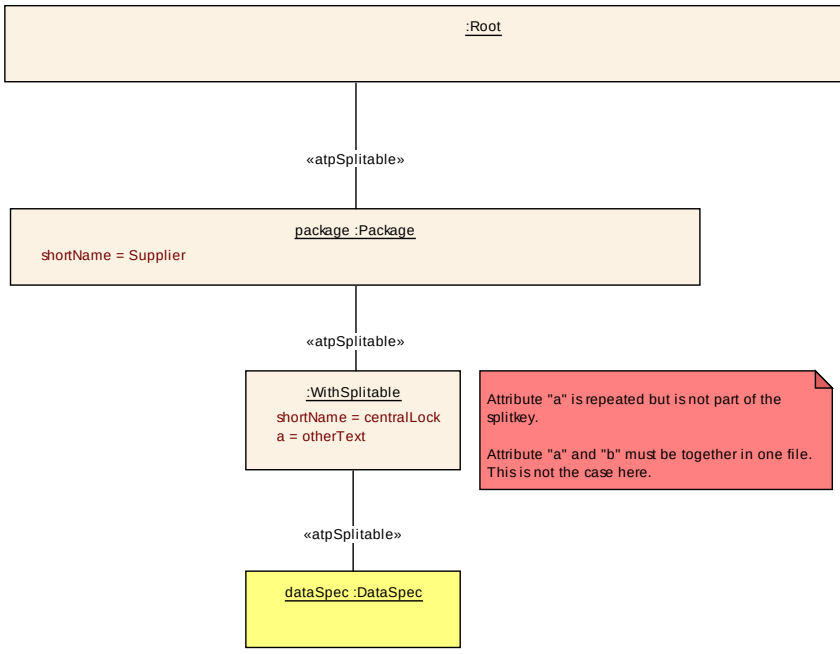

Figure 2.6: Invalid Partial Model (2)

2.4 UML Tags

The AUTOSAR Template Profile uses the following UML tags. Note that only those tags are mentioned here which directly influence the semantic content of the meta model.

- `atp.recommendedPackage`
  This tag provides a recommended package name for objects of the given meta

class. Thereby it provides a value for {kind} in Chapter 3.1. Usually it is the name of the meta class to which the tag is attached. Note that

  - this tag is propagated to subclasses
  - this tag only applies to subclasses of PackageableElement

- atp.Splitkey
  This specifies the identifying key of ≪atpSplitable≫ relations. For more details refer to 2.3.2. The tag specifies a comma separated list of OCL expressions. These expressions can be the foundation of constructing an identifying string.

  For example if in PhysicalChannel.iSignalTriggering atp.Splitkey is set to "shortName, variationPoint.shortLabel", one can generate the following OCL code:

```
context: PhysicalChannel
def: iSignalTriggering_atpSplitkey : String =
     self.iSignalTriggering.shortName.concat(',')
     .concat(self.iSignalTriggering.variationPoint.shortLabel)
```

- atp.Status
  This tag allows to specify the fact that the related entity is obsolete and kept in the meta model for backwards compatibility reasons. It is applicable to classes, aggregations, associations and attributes.

  The following values are supported:

  obsolete This indicates that related entity is obsolete and kept for compatibility reasons. If this tag is set, the note shall express the recommended alternative solution.

  preliminary This indicates that related entity is prelimiary in the model. It is subject to be changed without backwards compatibiilty management. An AUTOSAR release does not contain such elements. It is intended for AUTOSAR internal development.

  If the tag is not specified, the related entity is part of the current meta model.

- Vh.LatestBindingTime
  This tag controls the binding time of variant handling. For more details see chapter 7.6.

- xml.xsd ...
  These tags allow to define details of primitive types by using XSD restrictions. Even if it is specified by means of a technology specific definition it applies to the meta model independently of the storage technology.

- xml.xsd.customType
  This tag is applicable to a ≪primitive≫. It specifies the name of the xsd:simpleType which represents the primitive type.

- `xml.attribute`
  determines if the UML-attribute is serialized as an XML attribute. This tag allows to control the XML serialization and is not relevant for the semantics of an M1 model.

- `xml.attributeRef`
  determines if the UML-attribute is serialized as a reference to a global XML attribute. If set to `true` serializes the property as a reference to a global attribute. Applicable only if `xml.attribute` is set to `true`.

  The name of the referenced attribute is specified in `xml.name`. The namespace prefix of the referenced attribute is specified in `xml.nsPrefix`.

- `xml.enforceMinMultiplicity`
  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 in the standardized schema.

- `xml.enforceMaxMultiplicity`
  If true, enforce maximum multiplicity; otherwise, it is "unbounded". By default `xml.enforceMaxMultiplicity` is true.

- `xml.globalElement`
  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`
  determines the data type for a primitive if this is a primitive type generated by the meta model tool. Major example for such a generated type is given by `REFERRABLE-SUBTYPES-ENUM`. This tag shall be applied to a ≪`primitive`≫ which then acts as a proxy to the type denoted in `xml.mds.type`.

- `xml.name`
  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 [2].

- `xml.nsPrefix`
  This tag may be applied to

  - **attribute** determines the namespace prefix for properties with `xml.attributeRef` set to `true`
  - **package** determines the namespace prefix used for the schema based on this package.

- `xml.nsUri`
  determines the namespace URI used for the schema based on this package. The format of the namespace URI is defined in [2].
- `xml.roleElement`, `xml.roleWrapperElement`, `xml.typeElement`, `xml.typeWrapperElement`
  These tags allow to control the XML serialization and are not relevant for the semantics of an M1 model.
- `xml.sequenceOffset`
  determines the sequence in which the properties are serialized in XML. If this tag is missing, the properties are serialized in alphabetical order. This sequence is relevant for easier maintenance of the XML artifacts but are not relevant for the semantics of an M1 Model[6].
- `xml.systemIdentifier`
  determines the System identifier which should be used in instances dedicated to this schema.
- `msr.id`
  This is used to define subsets of the meta model when generating the XML schema and other artifacts. Based on this uml tag, and the use case configuration file, some meta-classes or relationships can be dropped. This further on allows to trace back to elements in the imported MSRSW.DTD. Note this is primarily for AUTOSAR internal use.

For document administration of the meta model, the UML tags listed below are applied to a particular package (for example `M2::AUTOSAR Templates::ReadMe`). If this package is referenced in the configuration files of the Meta Model Tool the values are forwarded to the generated artifacts (e.g. MMOD_XMLSchema or MOD_ECUConfigurationParameters).

In addition to the UML tags the disclaimer for the generated artifacts is taken from the package note of this package.

The following UML tags apply:

- `admin.documentClassification`
  denotes the classification of the meta-model (Standard resp. Auxillary)
- `admin.documentIdentificationNo`
  The AUTOSAR document number
- `admin.documentOwner`
  denotes the maintainer of the meta-model
- `admin.documentResponsibility`
  denotes the responsible authority of the meta-model

[6] If the sequence is relevant, this is denoted as {ordered} mutltiplicity in the model

- `admin.documentStatus`
  denotes the status of the meta-model
- `admin.documentTitle`
  denotes the title assigned to the meta-model
- `admin.documentVersion`
  denotes the official version of the meta-model. Note that the document version is not related to the `admin.partOfRelease` so a version number like 3.1.12 doesn't necessarily refer to the R3.1 branch of AUTOSAR.
- `admin.partOfRelease`
  denotes the AUTOSAR release in which the meta-model is published. Note that this tag is necessary as it's being used by the tooling in order to control the details of various generators such as:
  - the insertion of hardwired `xsd:simpleType` named `REF` for AUTOSAR release less than 4.0.
  - the processing of primitives according to Chapter 6.3.1.
  - structural differences of artifacts (e.g. classtables) among the AUTOSAR releases.
- `admin.releaseDate`
  denotes the date of the AUTOSAR release in which the meta-model is published
- `admin.revision`
  denotes the particular revision of the the AUTOSAR release in which the meta-model is published.

3 Autosar Top Level Structure

AUTOSAR uses a common top level structure for all AUTOSAR templates. This approach leaves maximum flexibility to design the artifacts in the AUTOSAR methodology. Figure 3.1 illustrates the AUTOSAR top level structure.

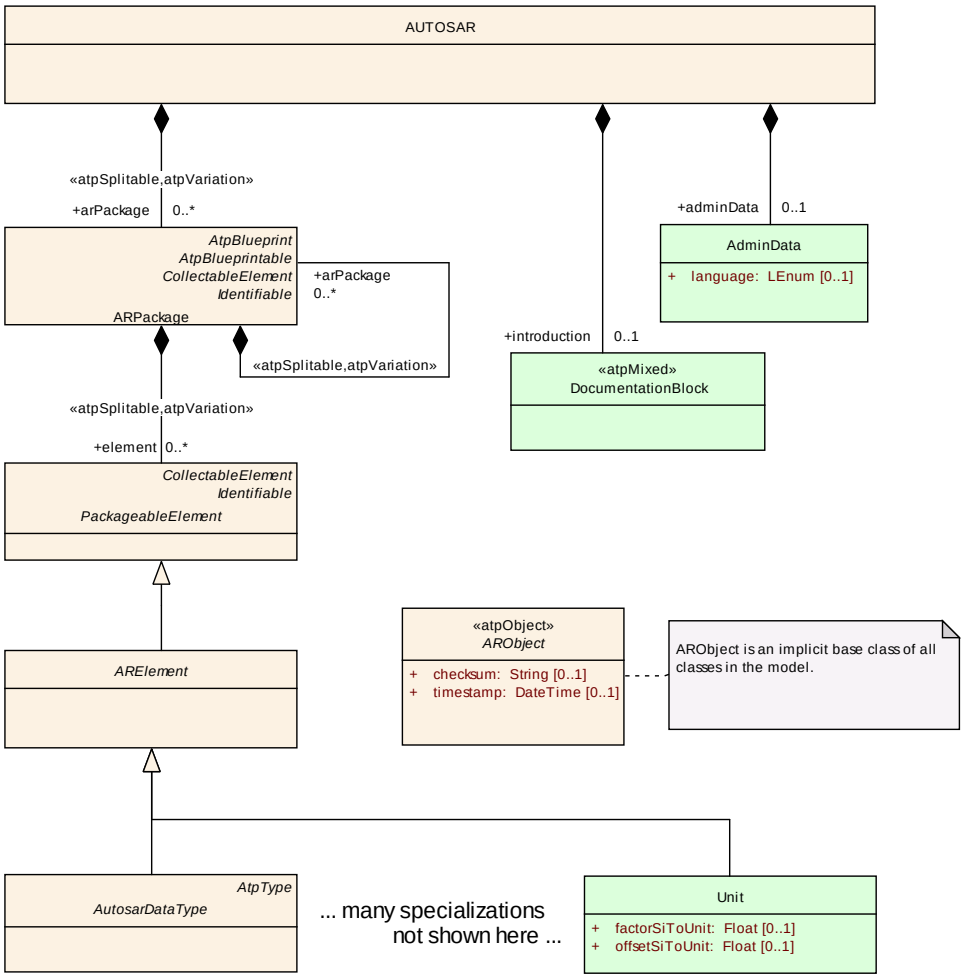

Figure 3.1: Top level structure of AUTOSAR templates

The meta-class `AUTOSAR` is the root of all templates. `AUTOSAR` contains multiple `ARPackage`s as `arPackage`. `ARPackage` can be arbitrarily nested. These packages contain `PackageableElement`s which represent particular autonomous entities of AUTOSAR templates. The most prominent specialization of this is `ARElement`.

Note that all[1] AUTOSAR meta-classes inherit from `ARObject` (see Chapter 4.1).

The top level structure also contains `AdminData` (see Chapter 4.5) which specifies two major aspects of an AUTOSAR artifact:

- change management information specified as `DocRevision`
- language status of the document

[1]Obviously `ARObject` dos not inherit from itself.

The language status of the artifact specifies:

1. the "master" language of the document specified in the attribute `language` in the top level `adminData`.
2. the additional languages which are in the document. This is specified as `usedLanguages` which is a `MultiLanguagePlainText` which serves as a list of languages used in the document. For more details see Chapter 8.6.

The following example illustrates the top-level structure of an ARXML file. This file is maintained in English as well as in German. English is the master language.

Listing 3.1: Top-Level Structure of an ARXML file

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://autosar.org/schema/r4.0"
      xsi:schemaLocation="http://autosar.org/schema/r4.0␣AUTOSAR_4-0-3.xsd"
         >
  <ADMIN-DATA>
    <LANGUAGE>EN</LANGUAGE>
    <USED-LANGUAGES>
      <L-10 L="EN" xml:space="default">English</L-10>
      <L-10 L="DE" xml:space="default">German</L-10>
    </USED-LANGUAGES>
  </ADMIN-DATA>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>demo</SHORT-NAME>
      <ELEMENTS>
        <!--
          autosar elements here
        -->
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

Class	**AUTOSAR**			
Package	M2::AUTOSARTemplates::AutosarTopLevelStructure			
Note	Root element of an AUTOSAR description, also the root element in corresponding XML documents. **Tags:** xml.globalElement=true			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data of an Autosar file. **Tags:** xml.sequenceOffset=10

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
arPackage	ARPackage	*	aggr	This is the top level package in an AUTOSAR model. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=BlueprintDerivation Time atp.Splitkey=shortName xml.sequenceOffset=30
introductio n	Documentation Block	0..1	aggr	This represents an introduction on the Autosar file. It is intended for example to rpresent disclaimers and legal notes. **Tags:** xml.sequenceOffset=20

Table 3.1: AUTOSAR

An AUTOSAR artifact is organized in `ARPackage`s which contains so called `PackageableElement`s elements. These are defined on its own nature, they exist independently from each other and are used by associations. For example a computation method is defined on its own. It is used by data definitions through a reference.

For more details about `ARPackage` please refer to Chapter 4.2.

3.1 Identifying M1 elements in packages

Packages are are used to organize AUTOSAR M1 models. AUTOSAR Gbr itself publishes M1 models as part of the released standard. In order to clearly identifying such model elements, the following rules apply:

- Model elements standardized by AUTOSAR and delivered as ARXML live in a top-level Package of which the `shortName` is `AUTOSAR`. This means that data elements which are defined by an OEM or supplier should **not** live in a top level package named `AUTOSAR`.

  The package structure follows the pattern:

```
AUTOSAR
    /{module}                                    -- identify the spec
       /{kind}s[_Blueprint | _Example]           -- identify the kind
                                                 --          of object
```

  An example structure is

```
/AUTOSAR
    /ComM
       /ApplicationDataTypes_Blueprint       [BLUEPRINT]
       /BswModuleEntrys_Blueprint            [BLUEPRINT]
       /CompuMethods_Blueprint               [BLUEPRINT]
       /DataConstrs_Blueprint                [BLUEPRINT]
       /DataTypeMappingSets_Blueprint        [BLUEPRINT]
```

```
/Documentations                          [STANDARD]
/ImplementationDataTypes_Blueprint       [BLUEPRINT]
/ImplementationDataTypes                 [STANDARD]
/ModeDeclarationGroups_Blueprint         [BLUEPRINT]
/SwcBswMappings_Blueprint                [BLUEPRINT]
/SwComponentTypes_Blueprint              [BLUEPRINT]
/BswModuleDescriptions_Blueprint         [BLUEPRINT]
```

In this example, there is a package with blueprints for implementation data types as well as for implementation data types which are finally implemented as STANDARD.

Another example is

```
/AUTOSAR
    /AISpecification
       /DataConstrs                          [STANDARD]
       /PhysicalDimensions                   [STANDARD]
       /Units                                [STANDARD]
       /ApplicationDataTypes_Blueprint       [BLUEPRINT]
       /CompuMethods_Blueprint               [BLUEPRINT]
       /PortInterfaces_Blueprint             [BLUEPRINT]
       /PortPrototypeBlueprints_Blueprint    [BLUEPRINT]
       /ApplicationDataTypes_Example         [EXAMPLE]
       /BlueprintMappingSets_Example         [EXAMPLE]
       /CompuMethods_Example                 [EXAMPLE]
       /PortInterfaces_Example               [EXAMPLE]
       /SwComponentTypes_Example             [EXAMPLE]
```

This example shows a use case which provides STANDARD, BLUEPRINT and EXAMPLE.

Note that that for compatibility reasons, the ECUC package structure is kept in AUTOSAR 4.0 as

```
/AUTOSAR
      /EcucDefs
```

- Model elements for which AUTOSAR specification already defines a standardized name (such as platform types) live in a top-level package of which the `shortName` is created according to the following pattern:

  `AUTOSAR_{module}[_{postfix}]/{kind}s`

In these given patterns, the following placeholders apply:

- `{module}` denotes a module designator

  [TPS_GST_0017] Module designator is ⌈

  – the module (as module abbreviation according to [6])

– defined by a keyword of classification `ModuleDesignator` in set "VirtualModules" specified in [7]

⌋

Basically it is one of:

`Adc, BswM, Can, CanIf, CanNm, CanSM, CanTp, CanTrcv, Com, ComM, CoreTst, Csm, DLT, Dbg, Dcm, Dem, Det, Dio, Ea, EcuM, Eep, Eth, EthIf, EthSM, EthTrcv, Fee, FiM, Fls, FlsTst, Fr, FrIf, FrNm, FrSM, FrTp, FrTrcv, Gpt, Icu, IpduM, J1939Tp, Lin, LinIf, LinNm, LinSM, LinTrcv, Mcu, MemIf, Nm, NvM, PduR, Port, Pwm, RamTst, Rte, SchM, SoAd, Spi, StbM, Ttcan, TtcanIf, UdpNm, Wdg, WdgIf, WdgM, Xcp`

`Crc, Bfx, Cal, E2E, Efx, Ifl, Mfl, Mfx, Ifx`

`AISpecification, Compiler, Comtype, EcuC, Std, Platform`

- `{postfix}` denotes the particular implementation in case that multiple implementations of the BSW Module appear in the same system.
- `{kind}` denotes the kind of element. The value is the name of a subclass of `ARElement`. The particular package names[2] are specified using the UML tag `atp.recommendedPackage` and shown as such in the class tables.

  Basically `{kind}` is one of

  `AliasNameSet, ApplicationDataType, BaseType, BlueprintMappingSet, BswModuleDescription, BswModuleEntry, CalibrationParameterValueSet, Collection, CompuMethod, ConstantSpecification, ConstantSpecificationMappingSet, DataConstr, DataTypeMappingSet, Documentation, EcucDefinitionCollection, EcucModuleConfigurationValues, EcucModuleDef, EcucValueCollection, EndToEndProtectionSet, EvaluatedVariantSet, FlatMap, HwCategory, HwElement, HwType, Implementation, ImplementationDataType, InterpolationRoutineMappingSet, KeywordSet, ModeDeclarationGroup, PhysicalDimension, PortInterface, PortInterfaceMappingSet, PortPrototypeBlueprint, PostBuildVariantCriterion, PostBuildVariantCriterionValueSet, PredefinedVariant, SwAddrMethod, SwAxisType, SwcBswMapping, SwComponentType, SwRecordLayout, SwSystemconst, SwSystemconstantValueSet, System, SystemSignal, SystemSignalGroup, TimingExtension, Unit, UnitGroup`

Packages have a `category` indicating if the elements in this package are used for reference purposes or if they are used for processing:

`BLUEPRINT` : Elements in such a package act as a kind of “blueprint” for real objects.

[2] Be aware of the plural ’s’

This applies in particular to objects such as PortInterface which are not modeled particularly to be a “blueprint”.

For example, an authoring tool provides the such predefined PortInterface as a kind of toolbox from which the definitions can be copied to a real project.

Model elements in such packages may be only defined partially. Therefore particular semantic constraints may apply.

[constr_2501] Blueprint of blueprints are not supported ⌈ Note that objects modeled particularly as a “blueprint” (e.g. PortBlueprint) also live in a package of category BLUEPRINT. Strictly speaking this means that they can be “blueprints” of “blueprints”. This indirection is not intended and not supported. ⌋

STANDARD : Elements which are standardized by the submitter of the related top level package and can be used as is for processing (e.g. ECU Parameter definitions).

Note that this also allows to represent stakeholder specific standard elements since STANDARD is not limited to AUTOSAR internal application.

ICS : Elements which form an Implementation Conformance Statement.

[constr_4055] ICS may not contain blueprints ⌈ Since an Implementation Conformance Statement always describes a set of one or more fully configured software modules, a package with category ICS it is not allowed to contain subpackages at any level which have the category BLUEPRINT. ⌋

For more details on the content of an Implementation Conformance Statement refer to [8].

EXAMPLE : Elements in EXAMPLE package are examples how to apply Elements in STANDARD or BLUEPRINT packages. These elements shall be ignored by generators etc.

Model elements which do not fall in to one of these categories should either live in a package with a category mutually agreed between the stakeholders. It is also possible to have no category at all in this case. **[constr_2515] Avoid conflicting package categories** ⌈ Note that it is in the responsibility of the stakeholders to ensure that no conflicting category occurs. ⌋

It is possible to maintain a reference from a blueprint to the “actual” objects which were derived from this blueprint. The meta-class BlueprintMappingSet can be used for this. Particular compatibility rules may be applicable and defined in the appropriate templates. For more details refer to [9].

Class	**BlueprintMappingSet**			
Package	M2::AUTOSARTemplates::StandardizationTemplate::BlueprintMapping			
Note	This represents a container of mappings between "actual" model elements and the "blueprint" that has been taken for their creation. **Tags:** atp.recommendedPackage=BlueprintMappingSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
blueprintMap	AtpBlueprintMapping	*	aggr	This represents a partiuclar blueprint map in the set.

Table 3.2: BlueprintMappingSet

3.2 The role of ARPackage, ARElement and Identifiable et. al.

The AUTOSAR meta model uses some abstract classes which represent various abilities with respect of model organization. A synopsis of the same is provided in Figure 3.2.

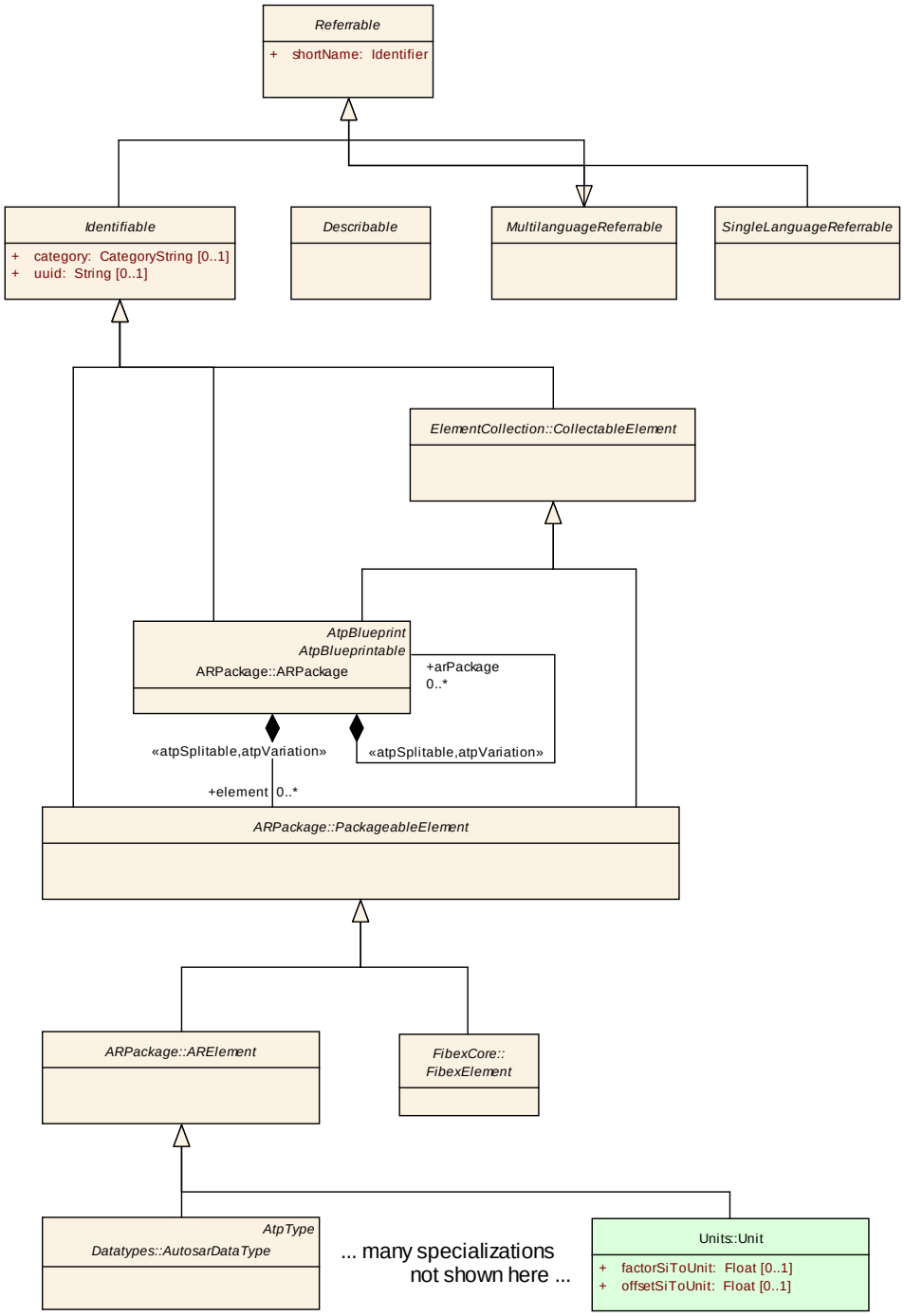

Figure 3.2: Synopsis of model organization classes

About `Referrable`:

- `Referrable` is an abstract meta class which has a `shortName` acting as a target for references. `Referrable` is mainly used for reference targets with a small footprint such as `Sdg`. `Referrable` is specialized in `SingleLanguageReferrable`/`MultiLanguageReferrable`.

- Specialization s of SingleLanguageReferrable are applicable as elements which are embedded in a text which is already part of a multilingual object (so called inline elements as described in Chapter 8.2.8).
- Specializations of MultiLanguageReferrable apply to multilingual objects which should have a relatively small footprint such as DefList, Traceable.

About Describable:

- Describable is an abstract meta class which represents the ability to provide a description but not being Referrable or Identifiable. It is mentioned here for completeness.

About Identifiable:

- Identifiable is an abstract class which inherits from MultilanguageReferrable. This is used to identify the essential objects in an AUTOSAR model. Related to Referrable it provides further means to identify the element such as desc. Obviously anything which can be identified shall also be referable. Therefore, Identifiable is a specialization of Referrable.
- Nested Identifiables establish a **hierarchical name space**.

  Note that if Referrable would contain further Referrables it would **not** be a hierarchical name space because name spaces are established by Identifiable only[3].

  See chapter 4.4.1 for more details.
- arPackage in AUTOSAR is the top-most Identifiable in an AUTOSAR model. This top-level ARPackage is the root of the name space hierarchy.
- The shortName of an Identifiable (more precisely of an Referrable) must be (case insensitively) unique in the containing Identifiable. Examples are
  - shortName of an ARElement (e.g. ApplicationSwComponentType) must be unique within ARPackage (which is Identifiable). But there might be other ApplicationSwComponentTypes with the same shortName in a **different** ARPackage.
  - The shortName of an PortPrototype must be unique within an ApplicationSwComponentType (which is Identifiable). But there might be PortPrototypes of the same name in other ApplicationSwComponentTypes.

About CollectableElement:

- CollectableElement represents the ability to be part of a collection (in particular to be referenced by a collection). This meta class does not introduce further attributes nor further possible aggregations.

[3]Nevertheless such a nesting of Referrables does not occur in the AUTOSAR meta model.

- Even if CollectableElement is similar to ARElement it is not the same because it handles an entirely different aspect.

About ARPackage:

- ARPackage can be nested: ARPackage can contain ARPackage in the role arPackage.
- Thereby ARPackage consists of nested branches (ARPackages) and leaves (subclasses of PackageableElement, in particular subclasses of ArElement).
- ARPackage is a specialization of Identifiable. Otherwise it would not contribute to the name space hierarchy.
- PackageableElement is an abstract class which represents the ability that the objects can be defined stand-alone. These objects do not require a context. Such objects are sometimes called "first class citizens".
- PackageableElement (ARElement FibexElement) cannot contain further PackageableElements (ARElements resp. FibexElement).

About ARElement and FibexElement:

- ARElement is an abstract class which contributes to an AUTOSAR model in general.
- FibexElement is an abstract class which represents the ability that the elements contribute particularly to the description of communication and topology of a system.
- ARElement and FibexElement is Identifiable. So the derived elements also have a shortName.
- ARElement FibexElement may contain further elements derived from Identifiable. An example for this is ApplicationSwComponentType.pPortPrototype which is of class PPortPrototype.

4 General Template Classes

The nature of the general template classes given below is similar to the standard library of a compiler: a set of predefined structures and elements to be used in an AUTOSAR template model.

4.1 ARObject - Common Attributes for all Classes

`ARObject` is one meta-class which is inherited by all other meta-classes according to the pattern shown in Figure 6.9.

Class	**ARObject (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ArObject			
Note	Implicit base class of all classes in meta-model.			
Base				
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
checksum	String	0..1	attr	Checksum calculated from the • attributes • aggregations and aggregated non-identifiables • references **Tags:** xml.attribute=true; xml.name=S
timestamp	DateTime	0..1	attr	The timestamp of the creation or modification of an instance, its attributes, references or aggregated non-identifiables. **Tags:** xml.attribute=true; xml.name=T

Table 4.1: ARObject

4.2 Packages in Autosar

AUTOSAR M1 models can be organized as a number of packages, represented by class `ARPackage`. It allows allows to put together the model elements, e.g. in form of an OEM or project specific package containing entities like a windshield wiper software component.

The self aggregation (role `arPackage`) shows that packages may in fact contain other packages. Besides those, a package may contain an arbitrary number of elements, represented by the abstract class `ARElement`. Such an AUTOSAR element is an entity for which it makes sense to be defined in its own semantic context (stand-alone). An example for such an `ARElement` is the definition of a reusable software component

type. On the other hand a parameter of an operation does not make sense to be defined stand-alone since its semantics is defined within and therefore highly dependent on the enclosing context: the operation.

The purpose of a package is to

- establish a name space for the elements in the package (as all `Identifiable`s do)
- can define the basis for relative references (see chapter 6.3.2.2)

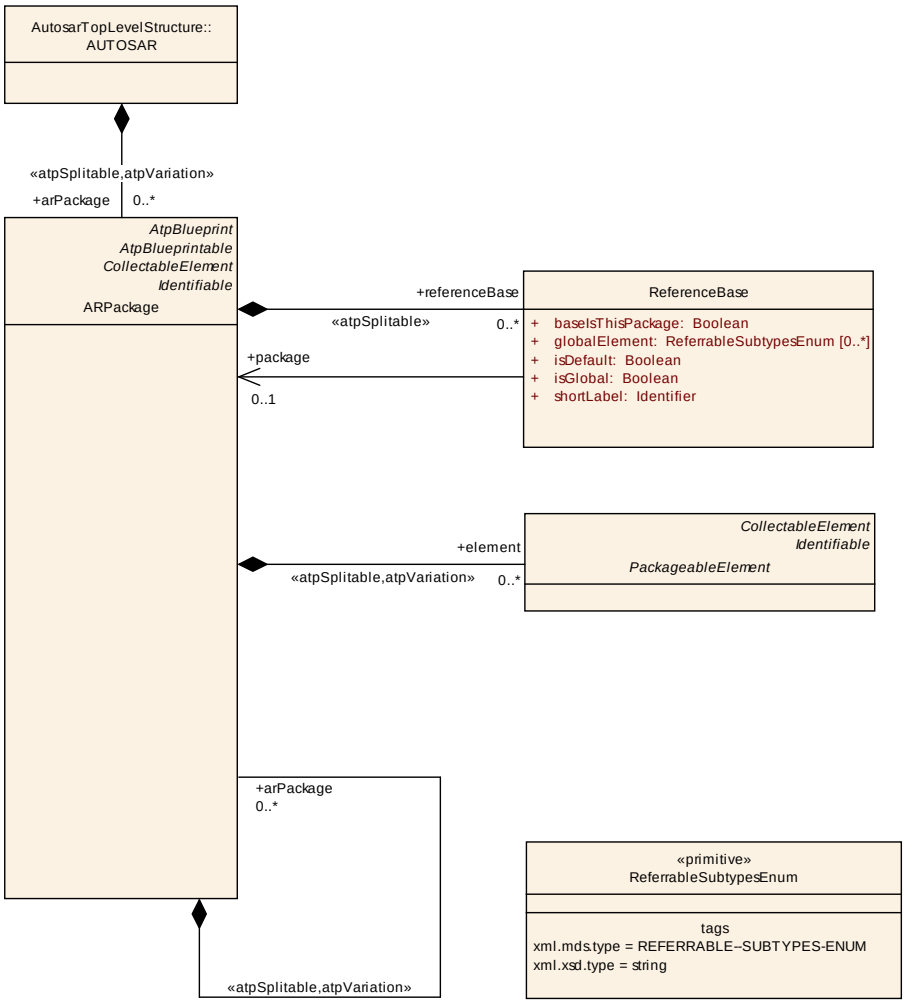

Figure 4.1: AUTOSAR package

Note that the aggregation of `ARElement` in `ARPackage` is subject to variation. The main use-case for this is to specify component alternatives with different interfaces within a product line architecture.

[constr_2537] Variation of `PackageableElement` is limited to components resp. modules ⌈ Variation of `ARElement` in `ARPackage` shall be applied only to elements on a kind of component level. In particular this is `BswModuleDescription`, `Documentation`, `Implementation`, `Documentation`, `SwComponentType`,

TimingExtension. This constraint only applies if the PackageableElement is not a blueprint. ⌋

[constr_2509] ReferenceBase needs to be unique in a package ⌈ The shortLabel of a reference base needs to be unique in (not within) a package. Note that it is not necessary to be unique within (to say in deeper levels) of a package. ⌋

[constr_2510] only one default ReferenceBase ⌈ Only one ReferenceBase per level can be marked as default (default="true"). ⌋

[constr_2538] Global reference is limited to certain elements ⌈ The ability to perform a global reference is limited to Chapter, Topic1, Caption, Traceable, XRefTarget, Std, XDoc, XFile ⌋

Class	**ARPackage**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	AUTOSAR package, allowing to create top level packages to structure the contained ARElements. ARPackages are open sets. This means that in a file based description system multiple files can be used to partially describe the contents of a package. This is an extended version of MSR's SW-SYSTEM.			
Base	ARObject,AtpBlueprint,AtpBlueprintable,Collectable Element,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
arPackage	ARPackage	*	aggr	This represents a sub package within an ARPackage, thus allowing an unlimited package hierarchy. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=BlueprintDerivation Time atp.Splitkey=shortName xml.sequenceOffset=30
element	PackageableElement	*	aggr	Elements that are part of this package **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=SystemDesignTime atp.Splitkey=shortName, variationPoint.shortLabel xml.sequenceOffset=20
referenceBase	ReferenceBase	*	aggr	This denotes the reference bases for the package. This is the basis for all relative references within the package. The base needs to be selected according to the base attribute within the references. **Stereotypes:** atpSplitable **Tags:** xml.sequenceOffset=10

Table 4.2: ARPackage

Class	**PackageableElement (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	This meta-class specifies the ability to be a member of an AUTOSAR package.			
Base	ARObject,CollectableElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 4.3: PackageableElement

Class	**ARElement (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	An element that can be defined stand-alone, i.e. without being part of another element (except for packages of course).			
Base	ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 4.4: ARElement

Class	**ReferenceBase**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	This meta-class establishes a basis for relative references. Reference bases are identified by the shortLabel which must be unique in the current package.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
baseIsThisPackage	Boolean	1	attr	This indicates that this base is established by the current package. In this case the association "package" can be derived as the qualified shortName of the enclosing package. If the value of baseIsThisPackage is set to true then one of the following must be true: • target of the association "package" must be the enclosing package. • association "package" is omitted. **Tags:** xml.sequenceOffset=28
globalElement	ReferrableSubtypesEnum	*	attr	This attribute represents a meta-class for which the global referencing is supported via this reference base. **Tags:** xml.sequenceOffset=29

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
isDefault	Boolean	1	attr	This attribute denotes if the current ReferenceBase is the default. Note that there can only be one default reference base within a package. **Tags:** xml.sequenceOffset=20
isGlobal	Boolean	1	attr	This indicates that the target of the applicable reference can be resolved via the non-qualified shortName. This requires that the shortName of the target is unique within the package referenced in the reference base. The default is false. Note that the reference base also maintains a list of elements which may be referenced using a "global Reference". **Tags:** xml.sequenceOffset=25
package	ARPackage	0..1	ref	This association specifies the basis of all relative references with the base equals shortLabel. This association must exist unless the value of baseIsThisPackage is set to true. **Tags:** xml.sequenceOffset=30
shortLabel	Identifier	1	ref	This is the name of the reference base. By this name, particular references can denote the applicable base. **Tags:** xml.sequenceOffset=10

Table 4.5: ReferenceBase

Primitive	**ReferrableSubtypesEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage
Note	This primitive is a proxy for an enum generated by the MMT. It allows to refer to any subclass of Referrable. Due to technical reasons the possible values are not shown in this class table. **Tags:** xml.mds.type=REFERRABLE–SUBTYPES-ENUM; xml.xsd.type=string

Table 4.6: ReferrableSubtypesEnum

4.3 Collections and Collectable Elements

For some use cases (e.g. evaluated variants (see Chapter 7.7) it is necessary to establish a collection of elements. Such collections are orthogonal to packages. Therefore

a collection resides in a package but is established by associations to the collected elements.

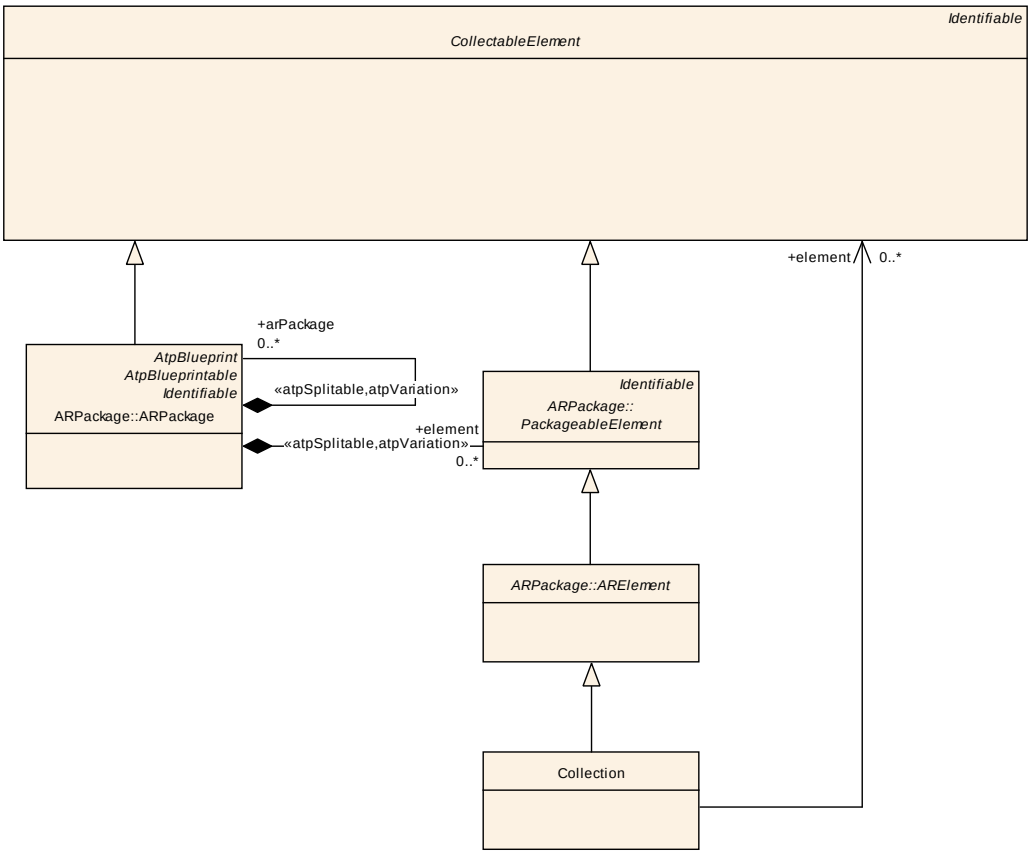

Figure 4.2: Collection of AUTOSAR elements

Class	**CollectableElement (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Element Collection			
Note	This meta-class specifies the ability to be part of an AUTOSAR collection.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 4.7: CollectableElement

Class	**Collection**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Element Collection			
Note	This meta-class specifies a collection of elements. A collection can be utilized to express additional aspects for a set of elements. Note that Collection is an ARElement. Therefore it is applicable e.g. for CertifiedVariant, even if this is not obvious. **Tags:** atp.recommendedPackage=Collections			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
element	CollectableElement	*	ref	This is an element in the collection. Note that Collection itself is collectable. Therefore collections can be nested.

Table 4.8: Collection

4.4 Identifiable and Referrable

The abstract base class `Identifiable` represents the ability to contribute significantly to the technical structure of an AUTOSAR system.

The abstract base class `Referrable` represents the ability to be the target of a reference. Such references can be simply links in a documentation or association contributing to the data model of an AUTOSAR system.

It is important to understand the big picture of the properties of `Identifiable`:

First, there are properties which have an identifying purpose:

- The `shortName` (provided by `Referrable`) unambiguously identifies the object within the context given by the first ancestor `Identifiable`.

  It is intended that `shortName`s are "speaking names" but can also be used for reference purposes.

- `longName` (provided by `Referrable`) which contains the headline of the object as **single line**. The content `longName` is targeted to human readers. Therefore the long name can be described in different languages.

- `desc` which contains a brief description of **what** the object is. This is intended to help a human being to identify the object. It is one **single paragraph** also provided in multiple languages.

- `adminData` contains administrative information about the object such as version information, language setting etc. This information also has identifying character.

In addition to these identifying properties, `Identifiable` carries the following properties which are of specifying purpose. For simplification of the model, these properties

are defined in `Identifiable` since they apply to mostly all identifiable objects representing the technical structure of an AUTOSAR system:

- `category` contains a keyword expressing a specific use case of the `Identifiable`. To some extent category can be compared with stereotypes in UML. The applicable categories are specified in the constraints of the objects in question.

  [TPS_GST_0016] Values for `category` ⌈In general it is allowed to extend the categories defined in the template specifications by user-defined values. In this case the user is responsible to avoid any conflict with existing or future defined AUTOSAR categories. This can be achieved for example by using an appropriate prefix.

  Anyhow the constraints of specific elements may restrict the category to exactly the defined ones and in this case an extension is not allowed. ⌋

- `introduction` contains introductory documentation about **how** the identified object is built or maybe used.

- `annotation` contains development annotations (see chapter 4.10).

As an example, these properties for the AUTOSAR project would be:

- shortName: AUTOSAR
- longName: AUTomotive Open Systems ARchitecture
- desc: AUTOSAR is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers.
- introduction:

  AUTOSAR

  - paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness
  - is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation"
  - ...

The base class `Identifiable` has further content related attributes, which are shown in figure 4.3:

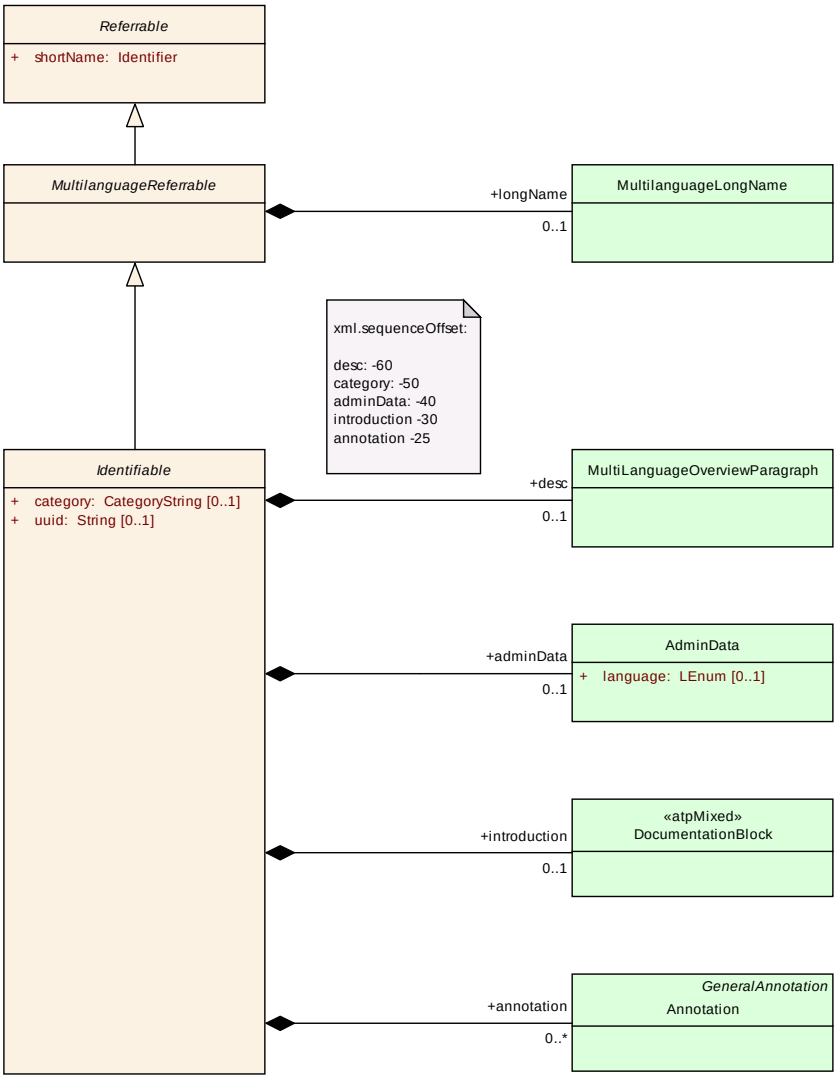

Figure 4.3: Identifiable

The base class `Referrable` and its specializations are shown in figure 4.4:

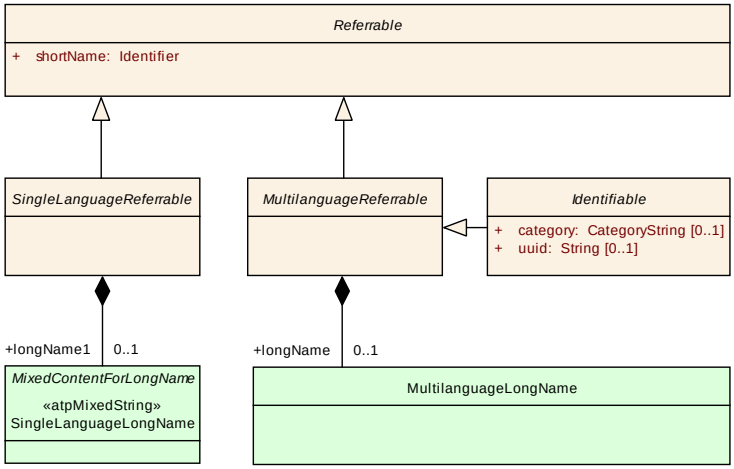

Figure 4.4: Referrable

For more details about `DocumentationBlock` please refer to chapter 8.2.

The attributes are given in the following class tables:

Class	**Identifiable (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.			
Base	ARObject,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
desc	MultiLanguage OverviewParagraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particlar how the object is built or used) should go to "introduction". **Tags:** xml.sequenceOffset=-60
category	CategoryString	0..1	attr	This element assigns a category to the parent element. The category is intended to specialize the usage and/or the content identifiable object. Such a specialization may also impose particular semantic constraints on the entire substructure (not only the identifiable itself). **Tags:** xml.sequenceOffset=-50
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Tags:** xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:** xml.sequenceOffset=-25
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:** xml.sequenceOffset=-30

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". **Tags:** xml.attribute=true

Table 4.9: Identifiable

Primitive	**Identifier**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes			
Note	An Identifier is a string with a number of constraints on its appearance, satisfying the requirements typical programming languages define for their Identifiers. This datatype represents a string, that can be used as a c-Identifier. It needs to start with a letter, may consist of letters, digits and underscore. It shall not have two consecutive underscores (to support subsequent name mangling based on "__"). **Tags:** xml.xsd.customType=IDENTIFIER; xml.xsd.maxLength=128; xml.xsd.pattern=[a-zA-Z]([a-zA-Z0-9]|_[a-zA-Z0-9])*_?; xml.xsd.type=string			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
namePattern	String	0..1	attr	This attribute represents a pattern which shall be used to define the value of the identifier if the identifier in question is part of a blueprint. For more details refer to TPS_StandardizationTemplate. **Tags:** xml.attribute=true

Table 4.10: Identifier

Class	**MultilanguageLongName**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::MultilanguageData			
Note	This meta-class represents the ability to specify a long name which acts in the role of a headline. It is intended for human readers. Per language it should be around max 80 characters.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
l4	LLongName	1..*	aggr	This is the long name in one particular language. **Tags:** xml.roleElement=true; xml.roleWrapperElement=false; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false

Table 4.11: MultilanguageLongName

Class	≪`atpMixedString`≫ **SingleLanguageLongName**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::SingleLanguageData			
Note	SingleLanguageLongName			
Base	ARObject,MixedContentForLongName			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 4.12: SingleLanguageLongName

Class	≪`atpMixedString`≫ **LLongName**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::LanguageDataModel			
Note	MixedContentForLongNames in one particular language. The language is denoted in the attribute l.			
Base	ARObject,LanguageSpecific,MixedContentForLongName			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 4.13: LLongName

Class	**MultiLanguageOverviewParagraph**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::MultilanguageData			
Note	This is the content of a multilingual paragraph in an overview item.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
l2	LOverviewParagraph	1..*	aggr	This represents the text in one particular language. **Tags:** xml.roleElement=true; xml.roleWrapperElement=false; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false

Table 4.14: MultiLanguageOverviewParagraph

Class	≪`atpMixedString`≫ **LOverviewParagraph**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::LanguageDataModel			
Note	MixedContentForOverviewParagraph in one particular language. The language is denoted in the attribute l.			
Base	ARObject,LanguageSpecific,MixedContentForOverviewParagraph			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 4.15: LOverviewParagraph

Class	**Referrable (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
shortName	Identifier	1	ref	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Tags:** xml.enforceMinMultiplicity=true; xml.sequenceOffset=-100

Table 4.16: Referrable

Class	**MultilanguageReferrable (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders). They also may have a longName. But they are not considered to contribute substantially to the overall structure of an AUTOSAR description. In particular it does not contain other Referrables.			
Base	ARObject,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
longName	MultilanguageLongName	0..1	aggr	This specifies the long name of the object. Long name is targeted to human readers and acts like a headline.

Table 4.17: MultilanguageReferrable

Class	**SingleLanguageReferrable (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders). They also may have a longName but in one language only. Specializations of this class only occur as inline elements in one particular language. Therefore they aggregate But they are not considered to contribute substantially to the overall structure of an AUTOSAR description. In particular it does not contain other Referrables.			
Base	ARObject,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
longName 1	SingleLanguage LongName	0..1	aggr	This specifies the long name of the object. The role is longName1 for compatibilty to ASAM FSX

Table 4.18: SingleLanguageReferrable

4.4.1 Name spaces and uniqueness of `shortName`

The `shortName` in `Referrable` contributes to the identification of specializations of `Referrable`, in particular `Identifiable`.

[TPS_GST_0018] Name Space established by `Identifiable` ⌈ `Identifiable` thereby establishes a name space in which the contained `Identifiable`s respectively `Referrable`s can unambiguously identified by their `shortName`. In some cases there are meta-classes derived from `Identifiable` that aggregate meta-classes that are not derived from `Identifiable` but they aggregate in turn a meta-class that **is** derived from Referrable. In other words, there are cases where `Identifiable`s are aggregated only indirectly.

Anyhow the rules for interpreting the established name space apply in this case as well. In particular, for a given instance of `Referrable`, the name space is established by the nearest **ancestor** (not only parent) element which is an `Identifiable`. This expressed by the phrase "within a given `Identifiable`" in [constr_2508]. ⌋

[TPS_GST_0019] `Referrable` does not establish Name Space ⌈ Note that `Referrable` does **not** establish a name space. Beyond `Identifiable` there is no use case where `Referrable` aggregates another `Referrable`. But if it would, the `shortName` of the aggregated one would need to be unique within the nearest ancestor `Identifiable` (not only the nearest ancestor `Referrable`). ⌋

Note that it is possible to extend the name space for particular elements in order to support global references. See [constr_2538] in Chapter 4.2 for more details.

[TPS_GST_0021] Case Sensitivity of `shortName` ⌈ `shortName` shall be

- basically interpreted as case sensitive (see [TPS_GST_0020])
- but checked for uniqueness as not case sensitive ([constr_2508])

⌋

[constr_2508] Name space of `shortName` ⌈ The content of `shortName` needs to be unique (case insensitive) within a given `Identifiable`.

Note that the check for uniqueness of `shortName` must be performed case insensitively. This supports the good practice that names should not differ in upper / lower case only which would cause a lot of confusion.

The term “case insensitive” indicates that the characters in the sets

```
{a b c d e f g h i j k l m n o p q r s t u v w x y z}
{A B C D E F G H I J K L M N O P Q R S T U V W X Y Z}
```

are respectively considered to be the same. In other words case-insensitive check for uniqueness of `shortName`s results in the fact that e.g. elements with `shortName` "X" and "x" are considered the same and shall **not** exist in the same package. ⌋

4.5 Administrative Data

`AdminData` is used to denote administrative meta data to objects. It covers various aspects:

- control of the languages in the document (see Chapter 8.6). The settings for multiple languages are specified in the top-Level `AdminData` only (see chapter 3).
- Version and change management is performed using `DocRevision`. Note that entry for the current revision is the first one. Information about previous revisions can be provided as change history.

  `DocRevision` allows to specify the revision label of its predecessors in order to document merge operations.

  `Sdg` allows to denote specific information in addition to the AUTOSAR standardized model. The usage of `Sdg` shall be mutually agreed between the involved parties.

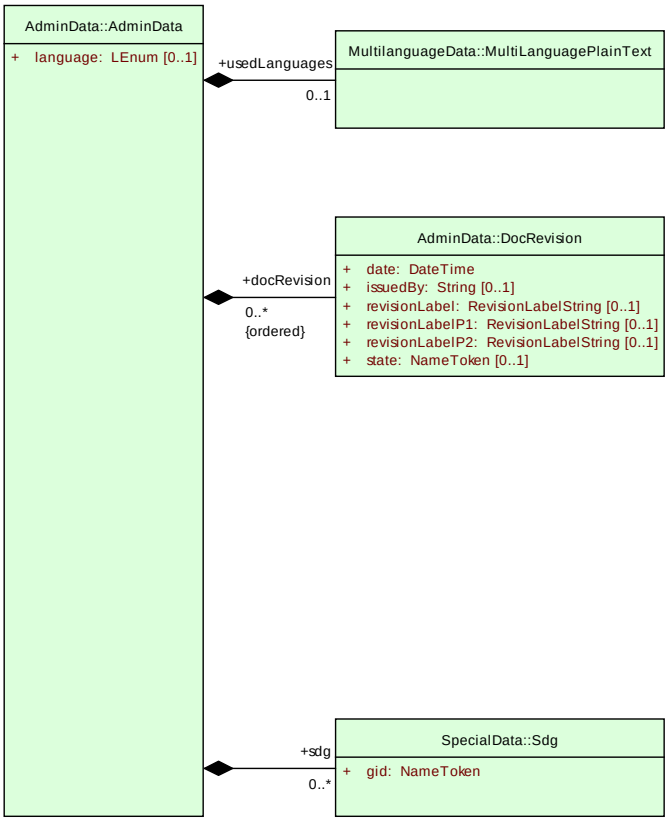

Figure 4.5: AdminData

Class	**AdminData**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::AdminData			
Note	AdminData represents the ability to express administrative information for an element. This administration information is to be treated as metadata such as revision id or state of the file. There are basically four kinds of metadata • The language and/or used languages. • Revision information covering e.g. revision number, state, release date, changes. Note that this information can be given in general as well as related to a particular company. • Document metadata specific for a company			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
docRevision (ordered)	DocRevision	*	aggr	This allows to denote information about the current revision of the object. Note that information about previous revisions can also be logged here. The entries shall be sorted descendent by date in order to reflect the history. Therefore the most recent entry representing the current version is denoted first. **Tags:** xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=50; xml.typeElement=false; xml.typeWrapperElement=false

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
language	LEnum	0..1	attr	This attribute specifies the master language of the document or the document fragment. The master language is the one in which the document is maintained and from which the other languages are derived from. In particular in case of inconsistencies, the information in the master language is priority. **Tags:** xml.sequenceOffset=20
sdg	Sdg	*	aggr	This property allows to keep special data which is not represented by the standard model. It can be utilited to keep e.g. tool specific data. **Tags:** xml.roleElement=true; xml.roleWrapper Element=true; xml.sequenceOffset=60; xml.type Element=false; xml.typeWrapperElement=false
usedLanguages	MultiLanguagePlainText	0..1	aggr	This property specifies the langauges which are provided in the document. Therefore it should only be specified in the top level admin data. For each language provided in the document there is one entry in MultilanguagePlainText. The content of each entry can be used for illustration of the language. The used langauge itself depends on the language attribute in the entry. **Tags:** xml.sequenceOffset=30

Table 4.19: AdminData

Class	**DocRevision**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::AdminData			
Note	This meta-class represents the ability to maintain information which relates to revision management of documents or objects. **Tags:** xml.sequenceOffset=20			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
date	DateTime	1	attr	This specifies the date and time, when the object in question was released **Tags:** xml.sequenceOffset=80
issuedBy	String	0..1	attr	This is the name of an individual or an organization who issued the current revision of the document or document fragment. **Tags:** xml.sequenceOffset=60
modification	Modification	*	aggr	This property represents one particular modification in comparison to its predecessor. **Tags:** xml.roleElement=true; xml.roleWrapper Element=true; xml.sequenceOffset=100; xml.type Element=false; xml.typeWrapperElement=false

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
revisionLabel	RevisionLabelString	0..1	attr	This attribute represents the version number of the object. **Tags:** xml.sequenceOffset=20
revisionLabelP1	RevisionLabelString	0..1	attr	This attribute represents the version number of the first predecessor of the object. **Tags:** xml.sequenceOffset=30
revisionLabelP2	RevisionLabelString	0..1	attr	This attribute represents the version number of the second predecessor of the object. This attribute is used if the object is the result of a merge process in which two branches are merged in to one new revision. **Tags:** xml.sequenceOffset=40
state	NameToken	0..1	attr	The attribute state represents the current state of the current file according to the configuration management plan. It is a NameToken until possible states are standardized. **Tags:** xml.sequenceOffset=50

Table 4.20: DocRevision

Class	**Modification**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::AdminData			
Note	This meta-class represents the ability to record what has changed in a document in comparison to its predecessor.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
change	MultiLanguageOverviewParagraph	1	aggr	This property denotes the one particular change which was performed on the object. **Tags:** xml.sequenceOffset=20
reason	MultiLanguageOverviewParagraph	0..1	aggr	This property represents the rationale for the particular change. **Tags:** xml.sequenceOffset=30

Table 4.21: Modification

4.6 Special Data

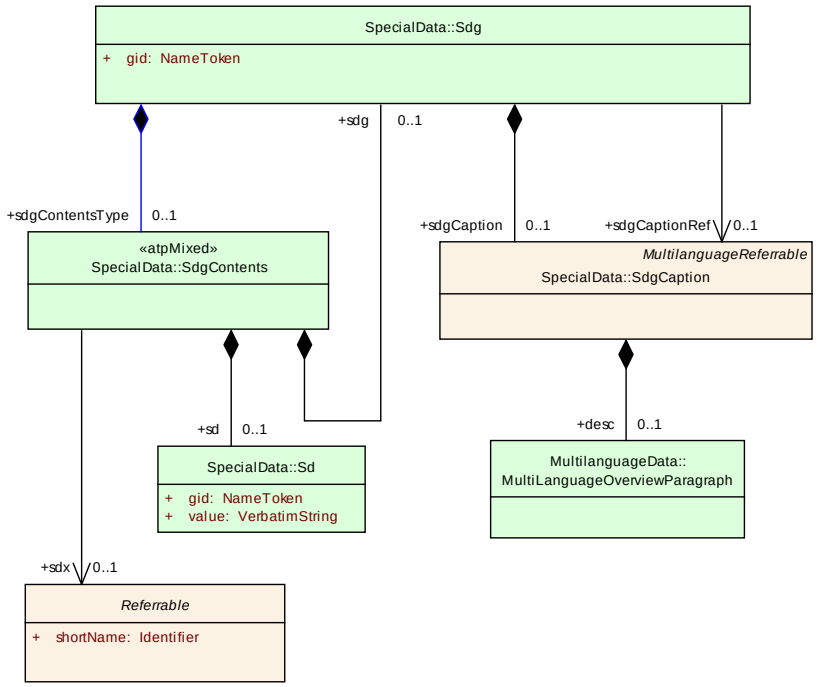

Figure 4.6: SpecialData

Class	**Sdg**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialData			
Note	Sdg (SpecialDataGroup) is a generic model which can be used to keep arbitrary information which is not explicitly modeled in the meta-model. Sdg can have various contents as defined by sdgContentsType. Special Data should only be used moderately since all elements should be defined in the meta-model. Thereby SDG should be considered as a temporary solution when no explicit model is available. If an sdgCaption is available, it is possible to establish a reference to the sdg structure.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
gid	NameToken	1	attr	This attributes specifies an identifier. Gid comes from the SGML/XML-Term "Generic Identifier" which is the element name in XML. The role of this attribute is the same as the name of an XML - element. **Tags:** xml.attribute=true
sdgCaption	SdgCaption	0..1	aggr	this aggregation allows to assign the properties of Identifiable to the sdg. By this, a shortName etc. can be assigned to the Sdg. **Tags:** xml.sequenceOffset=20

Attribute	Datatype	Mul.	Kind	Note
sdgCaptionRef	SdgCaption	0..1	ref	This association allows to reuse an already exisiting caption. **Tags:** xml.name=SDG-CAPTION-REF; xml.sequenceOffset=25
sdgContentsType	SdgContents	0..1	aggr	this is the content of the Sdg. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=30; xml.typeElement=false; xml.typeWrapperElement=false

Table 4.22: Sdg

Class	≪atpMixed≫ SdgContents			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialData			
Note	This meta-class represents the possible contents of a special data group. It can be an arbitrary mix of references, of primitive special data and nested special data groups.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
sd	Sd	0..1	aggr	This is one particular special data element. **Tags:** xml.sequenceOffset=40
sdg	Sdg	0..1	aggr	This aggregation allows to express nested special data groups. By this, any structure can be represented in SpeicalData. **Tags:** xml.sequenceOffset=50
sdx	Referrable	0..1	ref	Reference to any identifiable element. This allows to use Sdg even to establish arbitrary relationships.

Table 4.23: SdgContents

Class	SdgCaption			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialData			
Note	This meta-class represents the caption of a special data group. This allows to have some parts of special data as identifiable.			
Base	ARObject,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
desc	MultiLanguageOverviewParagraph	0..1	aggr	This represents a general but brief (one paragraph) description what the special data in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the special data in question.

Table 4.24: SdgCaption

Class	**Sd**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialData			
Note	This class represents a primitive element in a special data group.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
gid	NameToken	1	attr	This attributes specifies an identifier. Gid comes from the SGML/XML-Term "Generic Identifier" which is the element name in XML. The role of this attribute is the same as the name of an XML - element. **Tags:** xml.attribute=true
value	VerbatimString	1	attr	This is the value of the special data. **Tags:** xml.roleElement=false; xml.roleWrapper Element=false; xml.typeElement=false; xml.type WrapperElement=false

Table 4.25: Sd

4.7 Primitive Types

This chapter describes the primitive types which are used in the AUTOSAR M2 model. These primitives are shown in the class tables below. In addition to these primitives some packages may define some own local primitives.

Note that the AUTOSAR meta model does not use the built in primitives provided by UML.

Note further that some of these primitives also have attributes. Such attributes result in xml attributes of xml elements representing the primitive.

Enumeration	**ArgumentDirectionEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Primitive Types
Note	Use cases: • Arguments in ClientServerOperation can have different directions that need to be formally indicated because they have an impact on how the function signature looks like eventually. • Arguments in BswModuleEntry already determine a function signature, but the direction is used to specify the semantics, especially of pointer arguments.
Literal	***Description***
in	The argument value is passed to the callee.
inout	The argument value is passed to the callee but also passed back from the callee to the caller.
out	The argument value is passed from the callee to the caller.

Table 4.26: ArgumentDirectionEnum

Primitive	**Boolean**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	A Boolean value denotes a logical condition that is either 'true' or 'false'. It can be one of "0", "1", "true", "false" **Tags:** xml.xsd.customType=BOOLEAN; xml.xsd.pattern=0|1|true|false; xml.xsd.type=string

Table 4.27: Boolean

Enumeration	**ByteOrderEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	When more than one byte is stored in the memory the order of those bytes may differ depending on the architecture of the processing unit. If the least significant byte is stored at the lowest address, this architecture is called little endian and otherwise it is called big endian. ByteOrder is very important in case of communication between different PUs or ECUs.
Literal	***Description***
mostSignificantByteFirst	Most significant byte shall come at the lowest address (also known as BigEndian or as Motorola-Format)
mostSignificantByteLast	Most significant byte shall come highest address (also known as LittleEndian or as Intel-Format)
opaque	For opaque data endianness conversion has to be configured to Opaque. See AUTOSAR COM Specification for more details.

Table 4.28: ByteOrderEnum

Primitive	**CategoryString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This represents the pattern applicable to categories. It is basically the same as Identifier but has a different semantics. Therefore it is modeled as a primitive of its own. **Tags:** xml.xsd.customType=CATEGORY-STRING; xml.xsd.pattern=[A-Z][A-Z_]*; xml.xsd.type=string

Table 4.29: CategoryString

Primitive	**CIdentifier**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes			
Note	This datatype represents a string, that follows the rules of C-identifiers. **Tags:** xml.xsd.customType=C-IDENTIFIER; xml.xsd.pattern=[a-zA-Z_][a-zA-Z0-9_]*; xml.xsd.type=string			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
namePattern	String	0..1	attr	This attribute represents a pattern which shall be used to define the value of the identifier if the CIdentifier in question is part of a blueprint. For more details refer to TPS_StandardizationTemplate. **Tags:** xml.attribute=true

Table 4.30: CIdentifier

Primitive	**DateTime**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	A datatype representing a timestamp. The smallest granularity is 1 second. This datatype represents a timestamp in the format yyyy-mm-dd followed by an optional time. The lead-in character for the time is "T" and the format is hh:mm:ss. In addition, a time zone designator must be specified. The time zone designator can either be "Z" (for UTC) or the time offset to UTC, i.e. (+|-)hh:mm. Examples: 2009-07-23 2009-07-23T14:38:00+01:00 2009-07-23T13:38:00Z **Tags:** xml.xsd.customType=DATE; xml.xsd.pattern=([0-9]{4}-[0-9]{2}-[0-9]{2})(T[0-9]{2}:[0-9]{2}:[0-9]{2}(Z|([+\-][0-9]{2}:[0-9]{2})))?; xml.xsd.type=string

Table 4.31: DateTime

Primitive	**DisplayFormatString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes

Note	This is a display format specifier for the display of values e.g. in documents or in measurement and calibration systems. The display format specifier is a subset of the ANSI C printf specifiers with the following form: `% [flags] [width] [.prec] type character` For more details refer to "ASAM-HarmonizedDataObjects-V1.1.pdf" chapter 13.3.2 DISPLAY OF DATA. Due to the numerical nature of value settings, only the following type characters are allowed: • d: Signed decimal integer • i: Signed decimal integer • o: Unsigned octal integer • u: Unsigned decimal integer • x: Unsigned hexadecimal integer, using "abcdef" • X: Unsigned hexadecimal integer, using "ABCDEF" • e: Signed value having the form d.dddd e ddd where d is a single decimal digit, dddd is one or more decimal digits, ddd is exactly three decimal digits, and sign is + or - • E: Identical to the e format except that E rather than e introduces the exponent • f: Signed value having the form dddd.dddd, where dddd is one or more decimal digits; the number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision • g: Signed value printed in f or e format, whichever is more compact for the given value and precision; trailing zeros are truncated, and the decimal point appears only if one or more digits follow it • G: Identical to the g format, except that E, rather than e, introduces the exponent (where appropriate) **Tags:** xml.xsd.customType=DISPLAY-FORMAT-STRING; xml.xsd.pattern=%[\-+#]?[0-9]*(\.[0-9])?[diouxXfeEgGcs]; xml.xsd.type=string

Table 4.32: DisplayFormatString

Primitive	**Float**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	An instance of Float is an element from the set of real numbers. The value must comply with IEEE 754 and is limited to what can be expressed by a 64 bit binary representation. **Tags:** xml.xsd.customType=FLOAT; xml.xsd.type=double

Table 4.33: Float

Primitive	**Identifier**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes			
Note	An Identifier is a string with a number of constraints on its appearance, satisfying the requirements typical programming languages define for their Identifiers. This datatype represents a string, that can be used as a c-Identifier. It needs to start with a letter, may consist of letters, digits and underscore. It shall not have two consecutive underscores (to support subsequent name mangling based on "__"). **Tags:** xml.xsd.customType=IDENTIFIER; xml.xsd.maxLength=128; xml.xsd.pattern=[a-zA-Z]([a-zA-Z0-9]|_[a-zA-Z0-9])*_?; xml.xsd.type=string			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
namePattern	String	0..1	attr	This attribute represents a pattern which shall be used to define the value of the identifier if the identifier in question is part of a blueprint. For more details refer to TPS_StandardizationTemplate. **Tags:** xml.attribute=true

Table 4.34: Identifier

Primitive	**Integer**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	An instance of Integer is an element in the set of integer numbers (..., -2, -1, 0, 1, 2, ...). The value can be expressed in decimal, octal, hexadecimal and binary representation. Negative numbers can only be expressed in decimal notation Range is from -2147483648 and 2147483647. **Tags:** xml.xsd.customType=INTEGER; xml.xsd.pattern=[+\-]?[1-9][0-9]*|0x[0-9a-f]*|0[0-7]*|0b[0-1]*; xml.xsd.type=string

Table 4.35: Integer

Primitive	**MimeTypeString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes

Note	This primitive denotes the an Internet media type, originally called a MIME type after MIME and sometimes a Content-type after the name of a header in several protocols whose value is such a type, is a two-part identifier for file formats on the Internet. **Tags:** xml.xsd.customType=MIME-TYPE-STRING; xml.xsd.type=string

Table 4.36: MimeTypeString

Enumeration	**MonotonyEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This enumerator denotes the values for specification of monotony for e.g. curves.
Literal	***Description***
decreasing	This indicates that the related curve needs to be monotony decreasing.
increasing	This indicates that the related curve needs to be monotony increasing.
noMonotony	This indicates that the related curve needs not to be monotony.
strictlyDecreasing	This indicates that the related curve needs to be strictly monotony decreasing.
strictlyIncreasing	This indicates that the related curve needs to be strictly monotony increasing.

Table 4.37: MonotonyEnum

Primitive	**NameToken**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is an identifier as used in xml, e.g. xml-names. Basic difference to Identifier is the fact that it can contain "-". **Tags:** xml.xsd.customType=NMTOKEN-STRING; xml.xsd.type=NMTOKEN

Table 4.38: NameToken

Primitive	**NameTokens**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is a white-space separated list of name tokens. **Tags:** xml.xsd.customType=NMTOKENS-STRING; xml.xsd.type=NMTOKENS

Table 4.39: NameTokens

Primitive	**NativeDeclarationString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes

Note	This string contains a native data declaration of a data type in a programming language. It is basically a string, but white-space must be preserved. **Tags:** xml.xsd.customType=NATIVE-DECLARATION-STRING; xml.xsd.type=string; xml.xsd.whiteSpace=preserve

Table 4.40: NativeDeclarationString

Primitive	**Numerical**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive specifies a numerical value. It can be denoted in different formats such as Decimal, Octal, Hexadecimal, Float. See the xsd pattern for details. **Tags:** xml.xsd.customType=NUMERICAL-VALUE; xml.xsd.pattern=(0x[0-9a-f]*)|(0[1-7][0-7]*)|(0b[0-1]*)|([+\-]?[0-9]+(\.[0-9]*)?(E([+\-]?)[0-9]*)?); xml.xsd.type=string

Table 4.41: Numerical

Primitive	**PositiveInteger**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is a positive integer which can be denoted in Decimal, octal and hexadecimal. the value is between 0 and 4294967295. **Tags:** xml.xsd.customType=POSITIVE-INTEGER; xml.xsd.pattern=[1-9][0-9]*|0x[0-9a-f]*|0[0-7]*|0b[0-1]*; xml.xsd.type=string

Table 4.42: PositiveInteger

Primitive	**Ref**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes			
Note	This primitive denotes a name based reference. For detailed syntax see the xsd.pattern. • first slash (relative or absolute reference) • Identifier • a sequence of slashes and Identifiers This primitive is used by the meta-model tools to create the references. **Tags:** xml.xsd.customType=REF; xml.xsd.pattern=/?[a-zA-Z][a-zA-Z0-9_]{0,127}(/[a-zA-Z][a-zA-Z0-9_]{0,127})*; xml.xsd.type=string			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
base	Identifier	0..1	ref	This attribute reflects the base to be used for this reference. **Tags:** xml.attribute=true
index	PositiveInteger	0..1	attr	This attribute supports the use case to point on specific elements in an array. This is in particular required if arrays are used to implement particular data objects. The attribute shall b **Tags:** xml.attribute=true

Table 4.43: Ref

[constr_2552] Index attribute is only valid for arrays ⌈The index attribute in references is valid only if the reference target is an ArrayElement and/or has an attribute maxNumberOfElements.⌋

Primitive	**RegularExpression**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is a regular expression as defined in As of now it is still produced as a string in XSD. **Tags:** xml.xsd.customType=REGULAR-EXPRESSION; xml.xsd.type=string

Table 4.44: RegularExpression

Primitive	**RevisionLabelString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive represents a revision label which identifies an engineering object. It represents a pattern which • requires three integers representing from left to right MajorVersion, MinorVersion, PatchVersion. • may add an application specific suffix separated by one of ".", "_", ";". Legal patterns are for example: 4.0.0 4.0.0.1234565 4.0.0_vendor specific;13 4.0.0;12 **Tags:** xml.xsd.customType=REVISION-LABEL-STRING; xml.xsd.pattern=[0-9]+\.[0-9]+\.[0-9]+([\._;].*)?; xml.xsd.type=string

Table 4.45: RevisionLabelString

Primitive	**String**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This represents a String in which white-space must be normalized before processing. For example: in order to compare two Strings: • leading and trailing white-space needs to be removed • consecutive white-space (blank, cr, lf, tab) needs to be replaced by one blank. **Tags:** xml.xsd.customType=STRING; xml.xsd.type=string

Table 4.46: String

Primitive	**TimeValue**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive type is taken for expressing time values. The numerical value is supposed to be interpreted in the physical unit second. **Tags:** xml.xsd.customType=TIME-VALUE; xml.xsd.type=double

Table 4.47: TimeValue

Primitive	**UnlimitedInteger**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	An instance of UnlimitedInteger is an element in the set of integer numbers (..., -2, -1, 0, 1, 2, ...). The range is limited by constraint 2534. The value can be expressed in decimal, octal, hexadecimal and binary representation. Negative numbers can only be expressed in decimal notation. **Tags:** xml.xsd.customType=UNLIMITED-INTEGER; xml.xsd.pattern=[+\-]?[1-9][0-9]*|0x[0-9a-f]*|0[0-7]*|0b[0-1]*; xml.xsd.type=string

Table 4.48: UnlimitedInteger

[constr_2534] Limits of unlimited Integer ⌈ Practically `UnlimitedInteger` shall be limited such that it fits into 64 bit.

If a signed value is represented the min value can be down to -9223372036854775808 (0x800000000000000014) and the max value can be up to 9223372036854775807 (0x7FFFFFFFFFFFFFF).

If an unsigned value is represented the min value can be down to 0 and the max value can be up to 18446744073709551615 (`0xFFFFFFFFFFFFFFFF`. ⌋

[TPS_GST_2501] C ⌈ompatibility of numerical values (in particular `Float`, `Numerical`, `PositiveInteger`, `UnlimititedInteger`) is defined independent of the representation (float, integer.octal/hex/binary/decimal) as:

$v1$ and $v2$ are compatible if and only if $abs(v1 - v2) < epsilon$ ⌋

Primitive	**UriString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	A Uniform Resource Identifier (URI), is a compact string of characters used to identify or name a resource. **Tags:** xml.xsd.customType=URI-STRING; xml.xsd.type=string

Table 4.49: UriString

Primitive	**VerbatimString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive represents a string in which white-space needs to be preserved. **Tags:** xml.xsd.customType=VERBATIM-STRING; xml.xsd.type=string; xml.xsd.whiteSpace=preserve

Table 4.50: VerbatimString

4.8 Formula language

This chapter details the introduction of a general purpose formula language. The formula language can be used in different processing steps in the methodology, e.g. XML-processors, C preprocessor, Modeling tools.

4.8.1 Applying formula language

Until Release 3 the AUTOSAR artifacts could not express dependencies, i.e. calculate the value of one parameter based on other parameter values, or define values based on variant information. Each of these use cases is represented as a specification of the abstract meta-class `FormulaExpression` as shown in figure 4.7.

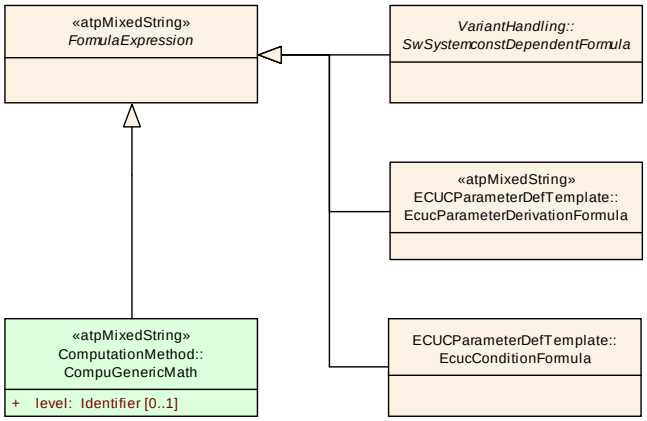

Figure 4.7: Formula language class hierarchy

The applicable operands are specified as associations in the subclasses. An example is given in figure 4.8. The valid `reference` in the grammar below are taken from the role names in the meta model. Maintaining the references as formal associations allows to retrieve dependencies even without parsing the formula expressions.

Please note that `FormulaExpression` is ≪`atpMixedString`≫ (see 2.3.1). Therefore one expression can be dependent on multiple `SwSystemconst`s.

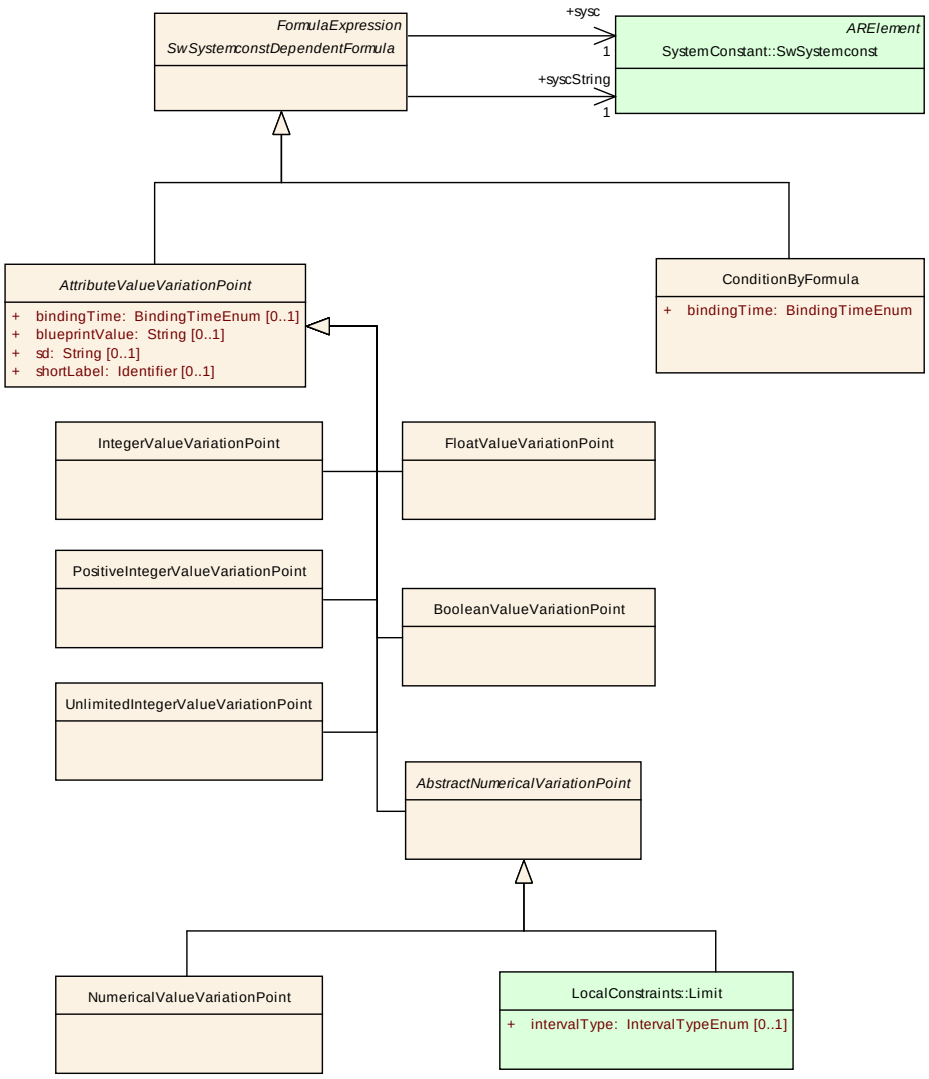

Figure 4.8: Formula depending on `SwSystemconst`

.

4.8.2 Formula language syntax

The formula language itself is compliant with C preprocessors respectively the programming language "'C"', the syntax is similar to the "C" language. The formula language uses the following syntax defined according to [10].

Note that the connection between the formula and referenced meta model objects is established by associations in the meta model and represented in the grammar as rule named `reference`.

Listing 4.1: AUTOSAR Formula language

```
grammar autosarFormulaLanguage;

expr
```

```
    :   atomExpr ;

atomExpr
    :   condExpr ;

condExpr
    :   orExpr (CondOperator orExpr AltOperator orExpr)? ;

orExpr
    :   xorExpr (OrOperator xorExpr)* ;

xorExpr
    :   andExpr (XorOperator andExpr)* ;

andExpr
    :   bitOrExpr (AndOperator bitOrExpr)*;

bitOrExpr
    :   bitXorExpr (BitOrOperator bitXorExpr)*;

bitXorExpr
    :   bitAndExpr (BitXorOperator bitAndExpr)*;

bitAndExpr
    :   compEqExpr (BitAndOperator compEqExpr)*;

compEqExpr
    :   compExpr (CompEqOperator compExpr)*;

compExpr
    :   shiftExpr (CompOperator shiftExpr)*;

shiftExpr
    :   sumExpr (ShiftOperator sumExpr)*;

sumExpr
    :   mulExpr (SumOperator mulExpr)*;

mulExpr
    :   powExpr (mulOperator powExpr)*;

powExpr
    :   unaryExpr (PowOperator unaryExpr)? ;

unaryExpr
    :   unaryOperator? atom;


atom
    :   DecIntegerLiteral
          | BooleanLiteral
          | HexIntegerLiteral
          | OctIntegerLiteral
          | BinIntegerLiteral
          | DecimalLiteral
          | DoubleLiteral
```

```
            | Epsilon
            | Undefined
            | reference
            | ArgumentOperand
            | DefinedFuncName LPAREN (reference | stringReference)  RPAREN
            | StringFuncName LPAREN stringArg COMMA stringArg RPAREN
            | BinaryFuncName LPAREN atomExpr COMMA atomExpr RPAREN
            | UnaryFuncName LPAREN atomExpr RPAREN
            | (LPAREN atomExpr RPAREN)
             ;

stringArg
   :   stringReference | StringLiteral;

unaryOperator
   :   SumOperator | '!' | '~' ;

//-------------------------------------------------------------------------
// these rules represent the reference to operands
// in an XML-Arttifact, this is represented as XML-Artifact
//
// Applicable RefFuncNames depend on the associations in the
// particular specialization of FormulaLanguage in the
// metamodel


stringReference
   :      ('syscString' | 'ecucQueryString') LPAREN  '/'? NCName ('/'
      NCName)* RPAREN;

reference
   :      ('sysc' | 'ecucQuery') LPAREN  '/'? NCName ('/' NCName)* RPAREN;




//
   ------------------------------------------------------------------------


// argumentOperand is valid only in formula derived from CompuGenericMath

ArgumentOperand
   :    'X' ('1' .. '9')('0' .. '9')* ;
// Tokens

OrOperator
   :   '||';

XorOperator
   :   '^^' ;

AndOperator
   :   '&&';
```

```
BitOrOperator
   :   '|';

BitXorOperator
   :   '^';

BitAndOperator
   :   '&';

CompEqOperator
   :   '==' | '!=';

CompOperator
   :   '<=' | '<' | '>=' | '>' | ;

ShiftOperator
   :   '<<' | '>>' ;

SumOperator
   :   '+' | '-';

// mulOperator is made a rule instead of a token due
// to antLr behavior. Otherwise '/' somehow collides with '/' in refString

mulOperator
   :   '*' | '/' | '%' ;

PowOperator
   :   '**' ;

CondOperator
   :   '?';

AltOperator
   :   ':';

DefinedFuncName
   :   'defined' ;

StringFuncName
   :   'streqcs' | 'streqci';

UnaryFuncName
   :    'round' | 'ceil' | 'floor' | 'abs' | 'log' | 'exp' |
        'sin' | 'cos' | 'asin' | 'acos' | 'atan' |
        'sinh' | 'cosh' | 'tan' | 'tanh' |
        'sqrt' | 'log10' |  'sgn'
        ;

BinaryFuncName
   :   'max' | 'min' | 'pow';

Epsilon
  : 'epsilon' ;

Undefined
```

```
  : 'undefined' ;


BooleanLiteral
    :  'true' | 'false' ;

LPAREN    :   '(' ;
RPAREN    :   ')' ;
COMMA     :   ',' ;


NCName : (Letter) (Letter | ('0'..'9') | '_')*;


fragment Letter
   :  '\u0024' | // '\u005f'|
      '\u0041'..'\u005a' | '\u0061'..'\u007a' |
      '\u00c0'..'\u00d6' | '\u00d8'..'\u00f6' |
      '\u00f8'..'\u00ff' | '\u0100'..'\u1fff' |
      '\u3040'..'\u318f' | '\u3300'..'\u337f' |
      '\u3400'..'\u3d2d' | '\u4e00'..'\u9fff' |
      '\uf900'..'\ufaff'
   ;

DecIntegerLiteral
       :   '0' | ('1'..'9')('0'..'9')*;

HexIntegerLiteral
      :   '0' ('x' | 'X') (('0'..'9') | ('a'..'f') | ('A'..'F'))+;

OctIntegerLiteral
     :   '0'('0'..'7')+;

BinIntegerLiteral
   :   '0' ('b' | 'B') ('0' ..'1')+;



DecimalLiteral
   :  ('0'? '.' ('0'..'9')+) |  (('1'..'9')('0'..'9')* ('.' ('0'..'9')*)) ;

DoubleLiteral
   : (('0'? '.' ('0'..'9')+) |  (('1'..'9')('0'..'9')* ('.' ('0'..'9')*)?))
     ('e' | 'E') ('+' | '-')? ('0'..'9')+;



StringLiteral : '"' ((~('"'|'\\') | '\\' ~('\\') | '\\\\')*) '"';

WS
   :   (' ' | '\t' | '\r' )* {$channel=HIDDEN;} ;
```

Please note the following:

- Note that the grammar above has one production rule (**reference**) which shall be considered as an example. It is defined to make the definition in the ANTLR complete. This rule represents the reference to a model element according to the specialized formula in the meta model. For more information about the implementation of these references refer to chapter 4.8.2.1.
- The semantics of the operators is shown in Table 4.51
- The supported mathematical functions are shown in Table 4.52

Symbol	Operator
!, ~	Negation (boolean, bit-wise)
**	Exponentiation
*, /, %	Multiplication, division, modulo
+, -	Addition, subtraction, sign
<<,>>	Bit-wise shift (left, right) note that the ends are filled with "0".
<, <=, >, >=	Comparison: less than, less than or equal to, greater than, greater than or equal to
==, !=	Comparison: equal, unequal to
&	Bit-wise AND
^	Bit-wise XOR
|	Bit-wise OR
&&	Boolean AND where operand value 0 -> false, others -> true
^^	Boolean XOR where operand value 0 -> false, others -> true
||	Boolean OR where operand value 0 -> false, others -> true

Table 4.51: Operators in arithmetic expressions

Function	Param.	Type of result	Meaning
round	1	integer	rounds positive and negative numbers to the nearest whole number. Note that when processing such expressions, that `round` is defined for the value ranges -2147483648 to +4294967295.
ceil	1	integer	rounds positive and negative numbers up to the next whole number. Note that when processing such expressions, that `ceil` is defined for the value ranges -2147483648 to +4294967295.
floor	1	integer	rounds positive and negative numbers down to the next whole number. Note that when processing such expressions, that `floor` is defined for the value ranges -2147483648 to +4294967295.
abs	1	like operand	absolute value
log	1	float	natural logarithm (base e)
log10	1	float	lograrithm base 10 - Provided for A2L 1.6
sqrt	1	float	square root - Provided for A2L 1.6
sin	1	float	sinus - Provided for A2L 1.6
asin	1	float	arcus sinus - Provided for A2L 1.6
cos	1	float	cosinus - Provided for A2L 1.6
acos	1	float	arcus cosinus - Provided for A2L 1.6
sinh	1	float	sinus hyperbolicus - Provided for A2L 1.6
cosh	1	float	cosinus hyperbolicus - Provided for A2L 1.6
tan	1	float	tangens - Provided for A2L 1.6
atan	1	float	arcus tangens - Provided for A2L 1.6

tanh	1	float	tangens hyperbolicus - Provided for A2L 1.6
exp	1	float	exponential function (base e)
defined	1	0 or 1	checks whether the reference given as the argument is defined
sgn	1	integer	signum, result is one of -1, 0, +1
max	2	depends on operands	finds the maximum value
min	2	depends on operands	finds the minimum value
pow	2	float	power(base, exp) - compute the power - provided for A2L 1.6
streqcs	2	0 or 1	compares two strings case sensitive
streqci	2	0 or 1	compares two strings case insensitive

Table 4.52: mathematical functions in arithmetic expressions

4.8.2.1 Implementation details of a Formula Processor

The following implementation details apply:

- **[TPS_GST_0001] Connection between Formula and Model Elements** ⌈ The formula language mentioned above has a production rule (**`reference`**) which needs to be specified further. This production indicates, that at this point, a reference to a model element needs to be resolved.

  There are various options to implement this. One example is to substitute the model reference by an equivalent textual representation such as

```
stringReference
   :      ('syscString' | 'ecucQueryString') LPAREN
                        '/'? NCName ('/' NCName)* RPAREN;

reference
   :      ('sysc' | 'ecucQuery') LPAREN  '/'? NCName ('/' NCName)* RPAREN;
```

  But however it is implemented, the formula needs to be loaded from the xml file with the arguments specified as model reference. It is not allowed to accept AUTOSAR artifacts which contain such a textual representation. ⌋

- **[TPS_GST_0015] result of reference/stringReference** ⌈
  - `reference` shall yield a numerical / boolean value.
  - `stringReference` shall yield a string value.

  ⌋

- **[TPS_GST_0002] aborting logical expressions** ⌈ The calculation of expressions with boolean AND or OR is aborted if the first operand has produced a result so that the total result cannot be changed anymore by the second operand (as in C). This behavior arises in expressions such as:

```
defined(sysc(SY_COUNT)) && sysc(SY_COUNT) > 1
```

since here, if `SY_COUNT` is not defined, the check for "`> 1`" is not carried out and an unwanted error message is thus avoided. ⌋

- **[TPS_GST_0003] true and false** ⌈ Like in C a integer "0" respectively floating point "0.0" is interpreted as false. Every other value is treated as true within boolean expressions.

  The language also provides the literals "`true`" yielding `1` respectively "`false`" yielding `0` to express literal boolean values in expressions. ⌋

- **[TPS_GST_0004] Priority of Operations** ⌈
  The priority rules of C++[1] apply and are modeled in the grammar: multiplication and division take precedence over addition and subtraction. The exponent operator (`**`) has priority over these other mathematical operators.

  The unary minus has greater precedence than all other operators. For a complete list of the priorities please refer to [11].

  Example:

  - `-2**3` becomes `-8 = (-2)**3`
  - `-log(2.718281828)**2` corresponds to `(-log(2.718281828))**2` and therefore is equal to `+1`.

  ⌋

- **[TPS_GST_0005] left-to-right evaluation** ⌈ Binary operators shall be evaluated from left to right (left-to-right associativity). For example 2 == 2 == 2 shall be evaluated as (2 == 2) == 2. ⌋

- **[TPS_GST_0006] Associativity of XOR** ⌈ Boolean XOR operator ^^ is an additional operator which has no counterpart in C. The result of a boolean XOR is "1" if one of the operands is interpreted as false and the other as true. The result of a boolean XOR is "0" if both operands have the same value independent of whether the value is true or false. Other than boolean AND (&&) and boolean OR (||), boolean XOR always evaluates both operands. This is because the result of a boolean XOR cannot be determined by only evaluating e.g. the first operand.

  Hint: In hardware circuits sometimes an XOR gate with more than two inputs is used. For such a hardware XOR gate the output is "1" if and only if one of the inputs is "1" and all other inputs are "0". The XOR operator within arithmetic expressions is a binary operator and therefore behaves different than hardware XOR gates. This means in particular that according to left-to-right associativity e.g.
  1 ^^ 1 ^^ 1 is interpreted as (1 ^^ 1) ^^ 1 which yields "1". ⌋

[1] Note that XOR is not defined in C++.

- **[TPS_GST_0007] Shift operation** ⌈
  When the shift operation is performed, the first or the last bit (depending on the direction of the shift) is filled with "0".

  Shift left means that the bits are shifted towards the higher values. Shift right means that the bits are shifted towards the lower values.

  Example:

  – `0b0001 << 1` returns `0b0010`

  – `0b1111 >> 1` returns `0b0111`

  – `0b1111 << 1` returns `0b1110`

  – `0b1111 << 2` returns `0b1100`

  ⌋

- **[TPS_GST_0008] Types in Formula Expressions** ⌈
  The type of an arithmetic expression is one of

  – Integer in the range -429496295 to +4294967295

  – Float (internally represented by double)

  It will be determined by the types of its sole parts. For the result type of a function or operand see 4.8.2.2. ⌋

- **[TPS_GST_0009] Keyword 'epsilon'** ⌈
  `epsilon` represents an implementation specific constant intended to support the comparison of float values. It represents the smallest increment which can be expressed by the given implementation.

  For example instead of comparing a float with zero, one should use

  ```
  abs(sysc(x)< epsilon) ? 0 : 1
  ```

  ⌋

- **[TPS_GST_0010] Keyword 'undefined'** ⌈
  `undefined` represents a sub term which is undefined. It is subject to be replaced in further process steps. The main purpose is to denote blueprints of expressions. The result of `undefined` is the same as an undefined operand. Usually it yields a runtime error. The following expressions hold true:

  ```
  defined(undefined) = false
  true && undefined = error
  false && undefined = false
  undefined = error
  ```

  And consequently for OR it is

  ```
  true || undefined = true
  false || undefined = error
  ```

⌋

- **[TPS_GST_0011] Functions `round`, `ceil`, `floor`** ⌈
  These act as 'Integer-Cast' with rounding. The following applies:

  - **`round`**
    rounds positive and negative numbers to the next whole number.

```
Ex. : round(4.4) = 4      round(-4.4) = -4
      round(4.7) = 5      round(-4.7) = -5
      round(4.5) = 5      round(-4.5) = -5
```

  - **`floor`**
    rounds positive and negative numbers down to the next whole number.

```
Ex. : floor(4.4) = 4      floor(-4.4) = -5
      floor(4.7) = 4      floor(-4.7) = -5
      floor(4.0) = 4      floor(-4.0) = -4
```

  - **`ceil`**
    rounds positive and negative numbers up to the next whole number.

```
Ex. : ceil(4.4)  = 5      ceil(-4.4)  = -4
      ceil(4.7)  = 5      ceil(-4.7)  = -4
      ceil(4.0)  = 4      ceil(-4.0)  = -4
```

  ⌋

- **[TPS_GST_0013] Function `defined`** ⌈ `defined(reference)` returns `1` if the reference passed as a parameter is defined. Note that for example `SwSystemconstValue` can be defined using a formula. Such a 'redefined' `SwSystemconst` is always treated as 'defined' even if its formula refers to an undefined `SwSystemconst` (See TURBO2 in the example below).

  `defined(stringReference)` yields `1` if the reference passed as parameter can yield a string. For example in case of `SystemConstDependantFormula` if the referenced `SystemConst` refers a `CompuMethod` of category `TEXTTABLE`.

  Example:

```
suppose
      sysc(ZYL_ZA) is set to  4
      sysc(ZYLZA2) is set to  sysc(ZYLZA)+2
      sysc(TURBO)  is left undefined
      sysc(TURBO2) is set to  sysc(TURBO)+2

then
      defined(sysc(ZYL_ZA))  yields 1
      defined(sysc(ZYL_ZA2)) yields 1
      defined(sysc(TURBO))   yields 0
      defined(sysc(TURBO2))  yields 1    (TURBO2 is defined, even if
                                         the referenced TURBO
                                         is undefined.)
```

An expression such as:

```
defined (SY_COUNT) && (sysc(SY_COUNT) > 1)
```

is permitted here since, if `SY_COUNT` is not defined, the check for "`> 1`" is not carried out and an unwanted error message is thus avoided. ⌋

- **[TPS_GST_0014] Error handling** ⌈
  Error messages shall be exposed by an evaluator
  - if the arithmetic expression is syntactically incorrect
  - with division by 0
  - if the definition range of a function is violated
  - if the function value of a function is outside the range which can be represented (for example, this applies to `floor(10E22)`)
  - if the evaluation of an operand fails (refer to Chapter 4.8.2.1, "aborting logical expressions")

  An error message does not occur if the range that can be represented is exceeded when using the basic arithmetic operators `'-', '+', '*', '/'` ⌋

4.8.2.2 Resulting Data Types of Formula Expressions

The following return types apply for operator/operands:

[TPS_GST_0034] Return Types for Additive 0perators ⌈

```
+ - / *
      Integer, Integer  -> Integer (this is an integer division)

      Integer, Float    -> Float
      Float, Integer    -> Float
      Float, Float      -> Float
```

⌋

[TPS_GST_0035] Return Types for Multiply Operators ⌈

```
**, pow
      Integer, Integer  -> Integer

      Integer, Float    -> Float
      Float, Integer    -> Float
      Float, Float      -> Float
```

⌋

[TPS_GST_0036] Return Types for Logical Operators ⌈

```
||, &&,^^, ==, <=, <, >, >=, !=
      Integer, Integer  -> Integer representing boolean (0: false,  1: true)
```

```
      Integer, Float    -> Integer representing boolean (0: false,  1: true)
      Float, Integer    -> Integer representing boolean (0: false,  1: true)
      Float, Float      -> Integer representing boolean (0: false,  1: true)
```

⌋

[TPS_GST_0037] Return Types for Bitwise Operators ⌈

```
|, &, ^, <<, >>         bit-wise operators always render Integer
                        with value >= 0
      Integer, Integer  -> Integer
      Integer, Float    -> Fault
      Float, Integer    -> Fault
      Float, Float      -> Fault
                         using an operand of type Float
                         or negative Integer leads to incorrect
                         arithmetic expression and an error message
```

⌋

[TPS_GST_0038] Return Types for Binary Function ⌈

```
min
      Integer, Integer  -> Integer

      Float, Integer    -> Float
      Integer, Float    -> Float
      Float, Float      -> Float

max
      Integer, Integer  -> Integer
      Integer, Float    -> Float
      Float, Integer    -> Float
      Float, Float      -> Float
```

⌋

[TPS_GST_0039] Return Types for Negation Operators ⌈

```
~, !
      Integer           -> Integer
      Float             -> Fault
                           using an operand of type Float
                           or negative Integer leads to incorrect
                           arithmetic expression and an error message
```

⌋

[TPS_GST_0040] Return Types for Define ⌈

```
defined                 -> Integer representing boolean (0: false,  1: true)
```

⌋

[TPS_GST_0041] Return Types for Unary Functions ⌈

```
log, log10, exp
      Integer           -> Float
```

```
      Float             -> Float

abs
      Integer           -> Integer
      Float             -> Float

sgn
      Integer           -> Integer (the result is one of -1, 0, 1)
      Float             -> Integer (the result is one of -1, 0, 1)

ceil, floor, round
      Integer           -> Integer
      Float             -> Integer
```

⌋

[TPS_GST_0042] Return Types for Keywords ⌈

```
epsilon                 -> Float

true, false             -> Integer representing boolean (0: false,  1: true)
```

⌋

4.8.2.3 Examples for the arithmetic expressions

Examples of correct arithmetic expressions are[2]:

```
01   <VF>1</VF>
02
03   <VF>2.0</VF>
04
05   <VF>1.5E5</VF>
06
07   <VF>2.0 * (1 + 2)</VF>
08
09   <VF><SYSC-REF DEST="SW-SYSTEMCONST">/S/SY_ZYLZA</SYSC-REF></VF>
10
11   <VF><SYSC-REF DEST="SW-SYSTEMCONST">/S/SY_ZYLZA</SYSC-REF>
12         + <SYSC-REF>/S/SY_TURBO</SYSC-REF></VF>
13
14   <VF>defined(<SYSC-REF DEST="SW-SYSTEMCONST">/S/SY_ZYLZA</SYSC-REF>)</VF>
15
16   <VF>1 / <SYSC-REF DEST="SW-SYSTEMCONST">/S/SY_ZYLZA</SYSC-REF></VF>
17
18   <SW-SYSCOND>
19      defined(<SYSC-REF DEST="SYSTEM-CONSTANT">SY_COUNT<SYSC-REF>)
19      &amp;&amp; <SYSC-REF DEST="SYSTEM-CONSTANT">SY_COUNT<SYSC-REF> &lt; 10
20   </SW-SYSCOND>
20
```

[2]Note that the example does not reflect the BINDING-TIME

Please note, that in the example on line 14 integer arithmetic is applied (the result will be equal to 0) if the number of cylinders `SY_ZYLZA` was also defined as an integer and we assume that there is more than one cylinder.

Please note, that the example in line 18 ff shows that markup characters ("<" and "&") need to be represented as XML entities when the formula is serialized as arxml.

4.9 EngineeringObject

While developing AUTOSAR based systems, it is necessary to refer to physical files. These files can be artifacts in which an AUTOSAR model is stored, but may also be source files, diagrams etc. AUTOSAR M1 models may need to refer to such files. It is required to keep AUTOSAR XML files independent of the physical layout of hard drives. Therefore, such a reference is abstracted as an `AutosarEngineeringObject`. This follows the approach in [12].

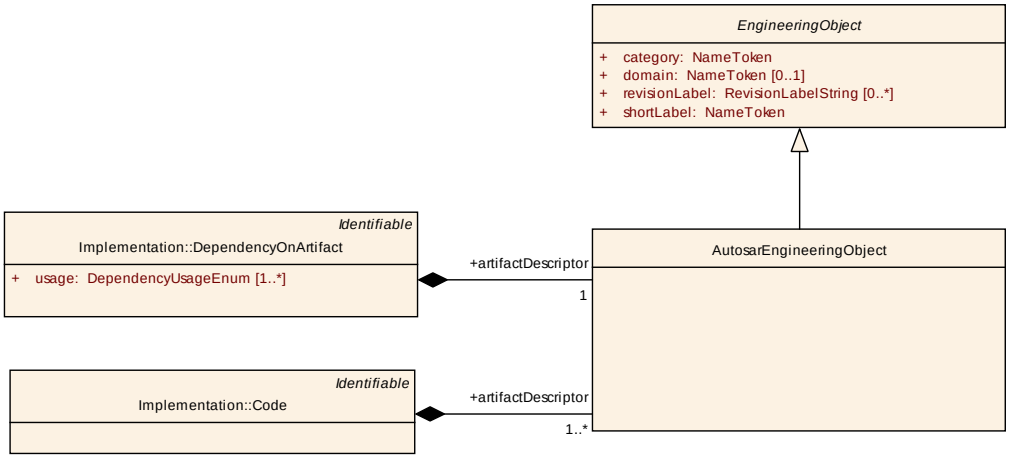

Figure 4.9: Engineering Object

Class	**EngineeringObject (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::EngineeringObject			
Note	This class specifies an engineering object. Usually such an object is represented by a file artifact. The properties of engineering object are such that the artifact can be found by querying an ASAM catalog file. The engineering object is uniquely identified by domain+category+shortLabel+revisionLabel.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
category	NameToken	1	attr	This denotes the role of the engineering object in the development cycle. Categories are such as • SWSRC for source code • SWOBJ for object code • SWHDR for a C-header file Further roles need to be defined via Methodology. **Tags:** xml.sequenceOffset=20
domain	NameToken	0..1	attr	This denotes the domain in which the engineering object is stored. This allows to indicate various segments in the repository keeping the engineering objects. The domain may segregate companies, as well as automotive domains. Details need to be defined by the Methodology. Attribute is optional to support a default domain. **Tags:** xml.sequenceOffset=40
revisionLabel	RevisionLabelString	*	attr	This is a revision label denoting a particular version of the engineering object. **Tags:** xml.sequenceOffset=30
shortLabel	NameToken	1	attr	This is the short name of the engineering object. Note that it is modeled as NameToken and not as Identifier since in ASAM-CC it is also a NameToken. **Tags:** xml.sequenceOffset=10

Table 4.53: EngineeringObject

Class	**AutosarEngineeringObject**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::EngineeringObject			
Note	This denotes an engineering object being part of the process. It is a specialization of the abstract class EngineeringObject for usage within AUTOSAR.			
Base	ARObject,EngineeringObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 4.54: AutosarEngineeringObject

The following example illustrates the usage of an `EngineeringObject` to refer to a physical file.

Listing 4.2: Example for an artifact description

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://autosar.org/schema/r4.0"
         xsi:schemaLocation="http://autosar.org/schema/r4.0 AUTOSAR_4-0-2.
            xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>demo</SHORT-NAME>
      <ELEMENTS>
        <SWC-IMPLEMENTATION>
          <SHORT-NAME>foo</SHORT-NAME>
          <REQUIRED-ARTIFACTS>
            <DEPENDENCY-ON-ARTIFACT>
              <SHORT-NAME>Foo</SHORT-NAME>
              <ARTIFACT-DESCRIPTOR>
                <SHORT-LABEL>FOO</SHORT-LABEL>
                <CATEGORY>SWSRC</CATEGORY>
                <DOMAIN>AUTOSAR</DOMAIN>
              </ARTIFACT-DESCRIPTOR>
              <USAGES>
                <USAGE>COMPILE</USAGE>
              </USAGES>
            </DEPENDENCY-ON-ARTIFACT>
          </REQUIRED-ARTIFACTS>
        </SWC-IMPLEMENTATION>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

This reference can be resolved via a container catalog as defined in [13]: The `artifactDescriptor` describes the artifact "FOO" which is of category "SWSRC". Using this information, the path to the physical file can be resolved via the following catalog example. There it is the first ABLOCK.

Listing 4.3: Example for an ASAM catalog

```
<?xml version = "1.0" encoding = "utf-8"?>
<CATALOG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="catalog_V3_0_0.ml.xsd">
  <SHORT-NAME>sample</SHORT-NAME>
  <ABLOCKS>
    <ABLOCK>
      <SHORT-NAME>FOO</SHORT-NAME>
      <CATEGORY>SWSRC</CATEGORY>
      <DOMAIN>AUTOSAR</DOMAIN>
      <FILES>
        <FILE>source/c/foo.c</FILE>
      </FILES>
    </ABLOCK>
    <ABLOCK>
      <SHORT-NAME>FOO</SHORT-NAME>
      <CATEGORY>SWCT</CATEGORY>
      <DOMAIN>AUTOSAR</DOMAIN>
      <FILES>
        <FILE>AUTOSAR/xml/foo.arxml</FILE>
      </FILES>
```

```
    </ABLOCK>
    <ABLOCK>
      <SHORT-NAME>FOO</SHORT-NAME>
      <CATEGORY>ECUC</CATEGORY>
      <DOMAIN>AUTOSAR</DOMAIN>
      <FILES>
        <FILE>AUTOSAR/ecuc/foo.ecucvalues.arxml</FILE>
      </FILES>
    </ABLOCK>
  </ABLOCKS>
</CATALOG>
```

4.10 Annotations

In the development process it is often required to place annotation (a kind of yellow pads) to the model. In order to support this in a generic way, the abstract meta-class `GeneralAnnotation` is applied and specialized according to the particular use case.

If no further attributes are required, the concrete meta-class `Annotation` is used.

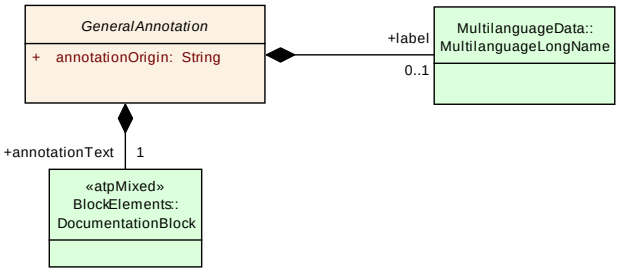

Figure 4.10: General Annotation

Class	**GeneralAnnotation (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::General Annotation			
Note	This class represents textual comments (called annotations) which relate to the object in which it is aggregated. These annotations are intended for use during the development process for transferring information from one step of the development process to the next one. The approach is similar to the "yellow pads" ... This abstract class can be specialized in order to add some further formal properties.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
annotation Origin	String	1	attr	This attribute identifies the origin of the annotation. It is an arbitrary string since it can be an individual's name as well as the name of a tool or even the name of a process step. **Tags:** xml.sequenceOffset=30

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
annotation Text	Documentation Block	1	aggr	This is the text of the annotation. **Tags:** xml.sequenceOffset=40
label	MultilanguageLongName	0..1	aggr	This is the headline for the annotation. **Tags:** xml.sequenceOffset=20

Table 4.55: GeneralAnnotation

Class	**Annotation**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::Annotation			
Note	This is a plain annotation which does not have further formal data.			
Base	ARObject,GeneralAnnotation			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 4.56: Annotation

4.11 MultiDimensionalTime

From timing point of view, it is important to specify a clear semantics for the timing properties (e.g. if a property has as unit seconds, or angular degrees). With the model element `MultidimensionalTime`, this can be done by using ASAM CSE code types (Codes for Scaling Units) as defined in citeASAM-MCD-2MC-ASAP2.

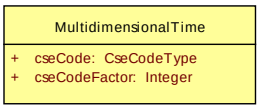

Figure 4.11: MultiDimensionalTime

Class	**MultidimensionalTime**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::MultidimensionalTime			
Note	This is used to specify a multidimensional time value based on ASAM CSE codes. It is specified by a code which defined the basis of the time and a scaling factor which finally determines the time value. If for example the the cseCode is 100 and the cseCodeFactor is 360, it represents 360 angular degrees. If the cseCode is 2 and the cseCodeFactor is 50 it represents 50 microseconds			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
cseCode	CseCodeType	1	attr	Specifies the time base by means of CSE codes.
cseCodeFactor	Integer	1	attr	The scaling factor for the time value based on the specified CSE code.

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Table 4.57: MultidimensionalTime

Primitive	**CseCodeType**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::MultidimensionalTime
Note	This primitive represents an ASAM CSE (Codes for Scaling Units) code as defined in the ASAM-MCD-2MC-ASAP2 specification. **Time** 0: 1 μsec 1: 10 μsec 2: 100 μsec 3: 1 msec 4: 10 msec 5: 100 msec 6: 1 sec 7: 10 sec 8: 1 min 9: 1 hour 10: 1 day **Angle** 100: Angular degrees 101: Revolutions (360 degrees) 102: Cycle (720 degrees) e.g. in case of IC engines **Cylinder Segment** 103: Cylinder segment Combustion e.g. in case of IC engines **others** 998: When frame available Time Source defined in the ASAP 2 keyword, FRAME 999: Always if there is new value Calculation of a new upper range limit after receiving a new partial value, e.g. when calculating a complex trigger condition 1000: Non deterministic Without fixed scaling **Tags:** xml.xsd.customType=CSE-CODE-TYPE-STRING; xml.xsd.type=unsignedInt

Table 4.58: CseCodeType

5 AbstractStructure

Abstract structures are used to define a kind of pattern which is applied by specialization. Abstract structures are established by

- abstract meta-classes
- relations between these meta-classes
  - marked as ≪atpAbstract≫ and/or ≪atpDerived≫
  - if it is atpDerived the target of the relation is marked as **derived**. This is shown in the diagrams by a slash preceding the role name.

    Derived means that the attribute is not directly in the model but somehow calculated from other information in the model. As an example, base in AtpInstanceRef is calculated as the container of the first atpContext.

    In consequence of this, derived relations do not appear in the XML schema. Specializations of abstract relations marked as derived must also be derived but not necessarily abstract. This allows to specify concrete derivations.

    Optionally the target of the relation can be marked as **derived union**. In this case the attribute is calculated as union of all concrete relations. This is shown in diagrams at the relation end in curly brackets. Note for such relations the upper multiplicity obviously needs to be greater than one.

Abstract structures are applied by

- subclasses the abstract meta-classes mentioned before.
- relations between these subclasses. These relations specialize the relationships between the abstract meta-classes. There are two kinds of specialization:
  - **redefines**
    redefine replaces the abstract relationship entirely
  - **subsets**
    subset contributes to the abstract relation such that it can be derived by building the union of all subsets.

  The specialization is shown in diagrams at the relation end in curly brackets. Note that relations of upper multiplicity equal 1 can only be “redefined” but not “subsetted”. On the other hand, relations with upper multiplicity greater than 1 can only be “subsetted” but not “redefined”.

Figure 5.1 illustrates the approach.

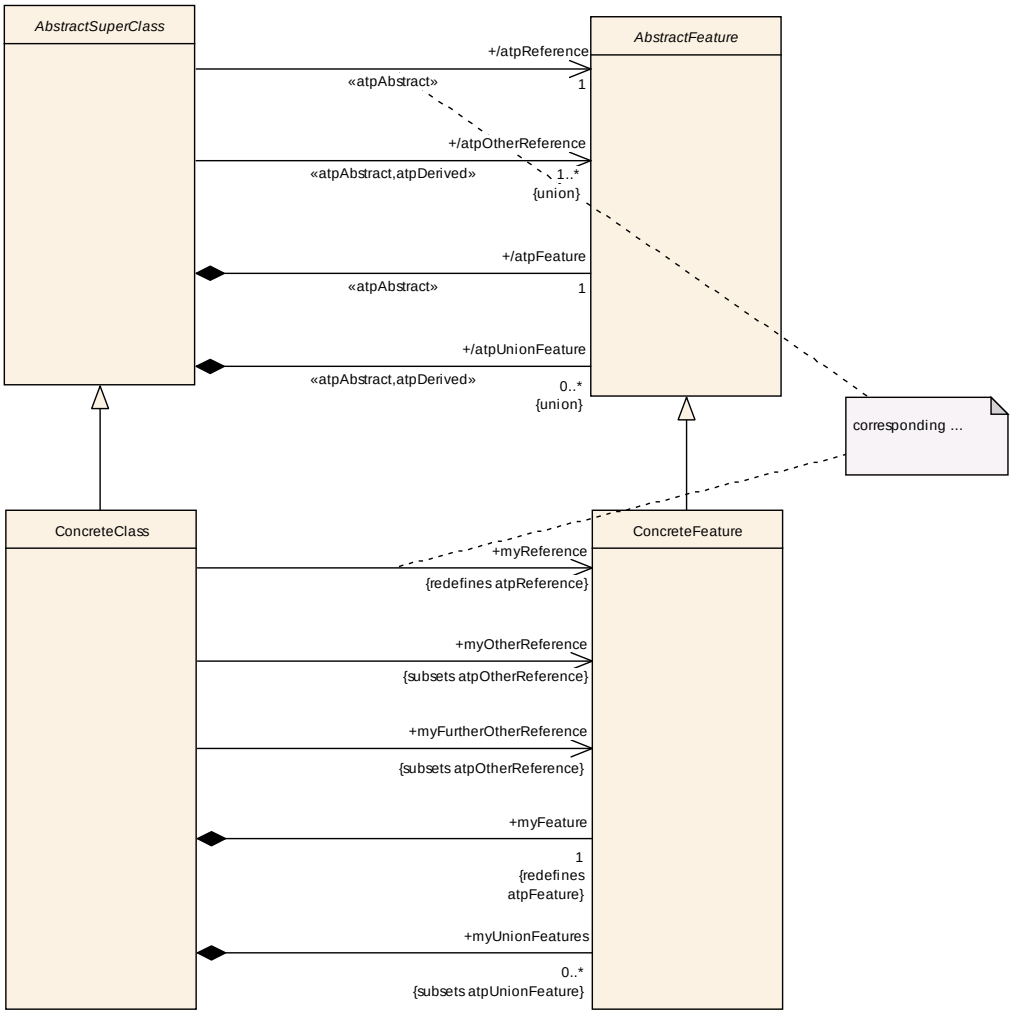

Figure 5.1: Definition and usage of abstract structures

5.1 Reusable Structural Hierarchies

5.1.1 Motivation

When designing a system it is often the case that elements in the runtime space share the same structure. A well-known example domain is object-oriented programming, where objects instantiated from the same class all have the same structure specified by that class. The ability to specify a structure once and then use it in multiple places in the design is also useful in the automotive domain. To account for this, the concepts of *types* and *prototypes* have been introduced into the AUTOSAR metamodel. A type represents a reusable structure and a prototype represents a use of such structure in a certain *role* within a type.

Consider the M1 model in Fig. 5.2. It shows an application component type ”Window-ControllerType” with a port prototype ”ctrl” typed by ”ControlInterface”, and a composition type ”PowerWindowType” which has two component prototypes by the names

”leftController” and ”rightController”, both typed by ”WindowControllerType”. Hence the type ”WindowControllerType” is used twice in the ”PowerWindowType” composition - once in the role of left and once in the role of right controller. Note that though the port ”ctrl” appears graphically twice within ”PowerWindowType” it is in fact specified only once, as part of the structure of ”WindowControllerType”.

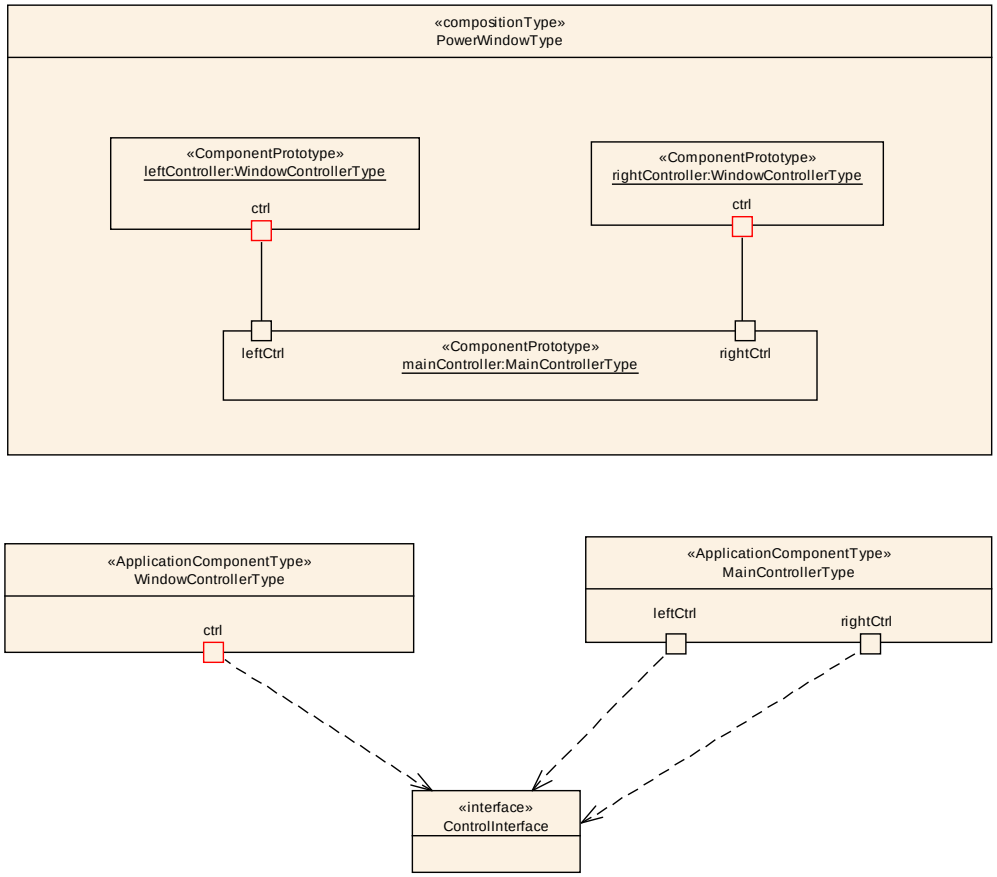

Figure 5.2: Reusable type example

The concept of reusable types results in a situation where a flat M1 model specifies deep, tree-like M0 instances. The structure of M0 instances of ”PowerWindowType” is (partly) shown in 5.3. As can be seen, there are two instances corresponding to the ”ctrl” port, defined once in M1.

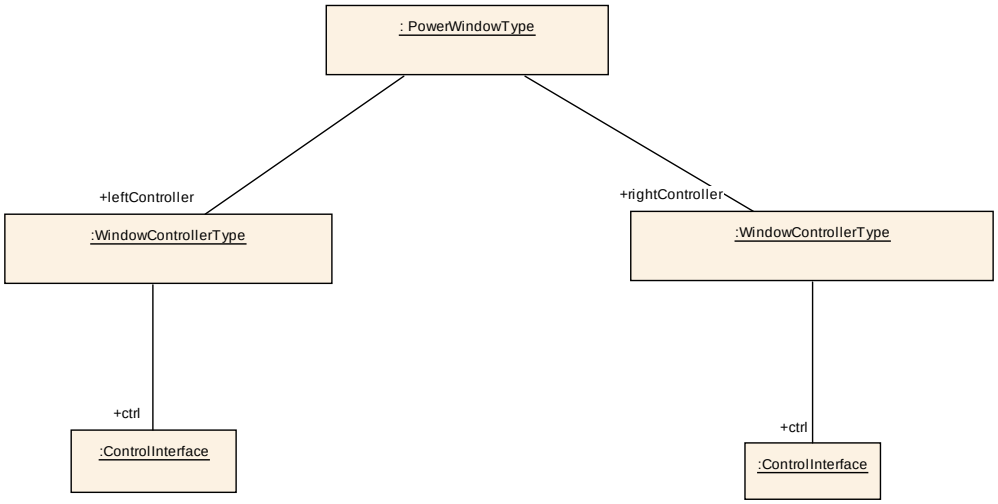

Figure 5.3: M0 instances of reusable types

It turns out that this duplication of structure in different roles also has consequences in the level of M1 models. Returning to Fig. 5.2, the ”PowerWindowType” composition also contains a component prototype ”mainController” typed by ”MainControllerType” which has left and right control ports. The connectors inside the composition connect the port of the left window controller to the left port of the main controller and the port of the right window controller to the right port of the main controller.

Recall that though the ”ctrl” port of the left and right controller appear graphically twice in the figure they in fact appear only once in the M1 specification - in the type ”WindowControllerType”. But in order to well-define the connector (e.g. in the XML description) there must be a way to distinguish *in the M1 model specification* between those two future M0 instances. This is because we need to attach the left instance to the ”leftCtrl” port of the main controller and the right to the ”rightCtrl” port. So the problem is how to refer to distinct would-be M0 instances which originate from the same M1 model element. This is addressed by the concept of *instance refs*.

The next section introduces the abstract layer for types, prototypes, and structure elements, and provides a more detailed account of these concepts. The next one introduces the the abstract layer for instance refs and explains this concept in more detail.

5.1.2 Types, Prototypes and Structure elements

Figure 5.4 shows the abstract layer for elements with internal structure. A *classifier* classifies instances according to their *features*. Here a ”classifier” means an M1 instance of (a concrete subclass of) the M2 meta-model class `AtpClassifier`, and ”features” are instances of (concrete subclasses of) the M2 meta-model class `AtpFeature`.

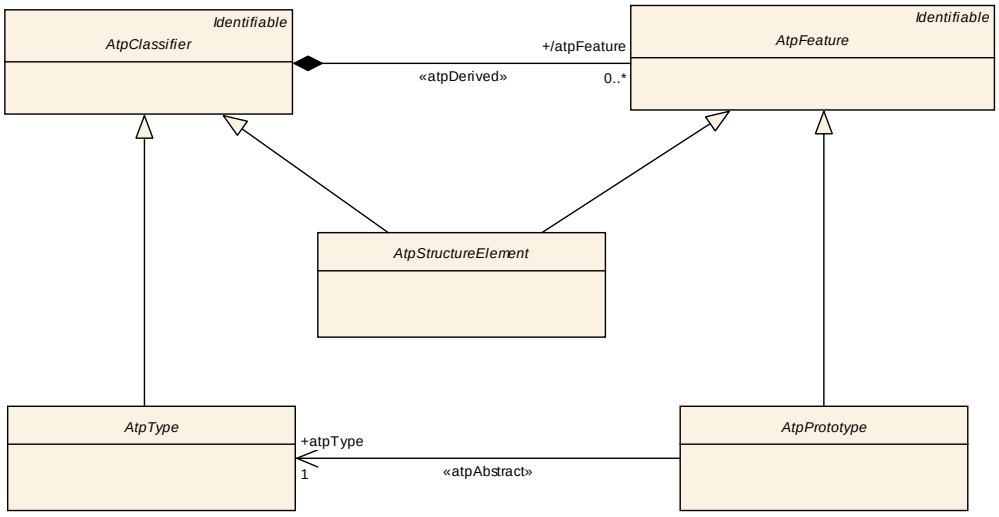

Figure 5.4: Abstract structure

An example of a classifier is some given component type and an example of a feature is some given port. So for example, the component type ”WindowControllerType” from Fig. 5.2 has a feature ”ctrl”, which is a port through which control signals arrive. Those elements are M1 elements and what they do is characterize M0 elements in the ”runtime” system. So the runtime system will have several instances of ”WindowControllerType” (two in each instance of ”PowerWindowType”) each of which will have a control port.

The set of all M0 instances of a given system may be partitioned, or classified, according to the features of each instance. A classifier represents an assembly of such features into a meaningful whole. Thus the M2 meta-class `AtpClassifier` is a ”vertical” concept in that its semantics, or meaning, cuts through layers of abstraction: the meaning of this M2 class is that M1 instances of it classify the M0 instance space.

The interplay of classifiers and features is such that the way by which a feature contributes to the specification of the classifier of which it is a part is via another classifier which specifies the structure of the feature. The meta-class `AtpPrototype` stands for features whose structure is given by another classifier, which types them. The meta-class `AtpType` stands for classifiers which type prototypes.

Some classifiers do not need to be reusable. This case is captured by the concept of *structure elements*. A structure element is a feature which is also a classifier and hence specifies its own structure instead of referencing to a type. The abstract class for this kind of element is `AtpStructureElement`. Structure elements are simpler to define because a single element does the job of both type and prototype.

Both types and structure elements are classifiers, i.e. have M0 instances. The difference is that types are reusable within an M1 model: a given type, e.g. ”WindowControllerType”, may be used to type many prototypes within a given model. Those prototypes represent different *roles* that instances of ”WindowLifterType” - window controller components - play in the containing composition. For example, one instance may play the role of ”leftLifter” and the other of ”rightLifter”.

The meta-classes `AtpType`, `AtpPrototype`, and `AtpStructureElement` are abstract. They are used in the meta-model as parent classes for concrete meta-classes. Figure 5.5 shows an example where composition types are defined as containers of component prototypes which in turn are typed by component types.

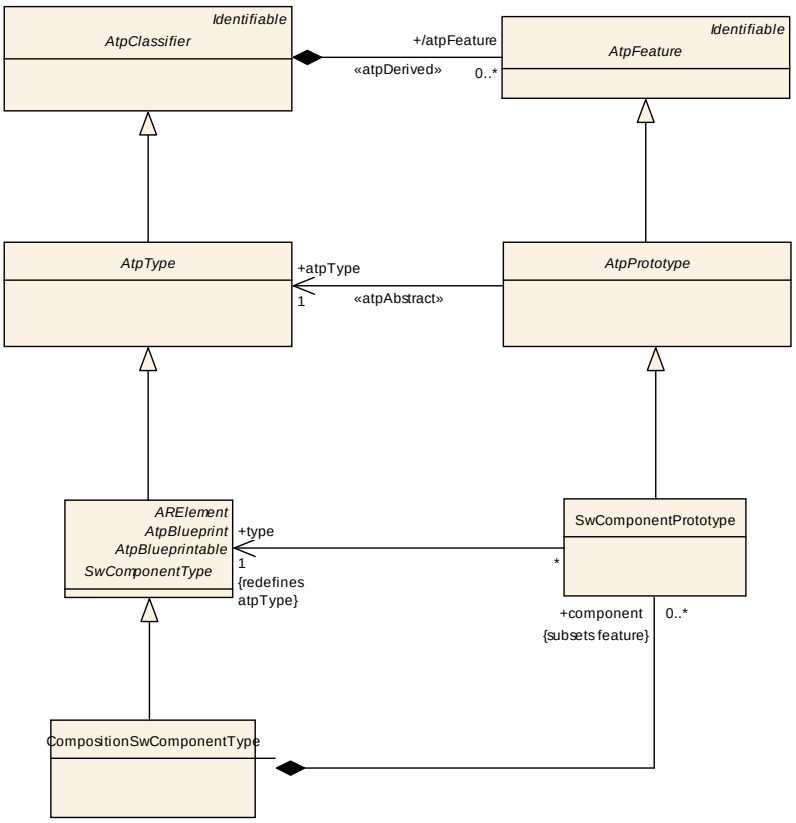

Figure 5.5: Concrete type-prototype

The structure of components as shown in the figure is a specialization of the general structure shown in figure 5.4. In addition to specializing the classes, the associations roled ”atpType” and ”atpFeature” are also specialized. ”atpType is redefined whereas ”atpFeature” is subsetted, in accordance with the fact that the first is abstract and the latter is a derived union. The concrete type association redefines the abstract atpType one. The union of the concrete feature associations (notice the plural) results in the derived atpFeature. So for example the features of a given component type include all its component prototypes **and** its ports. For technical reasons, the subsetting of features is not indicated in the meta-model diagrams.

Figure 5.6 shows an example of a concrete structure element.

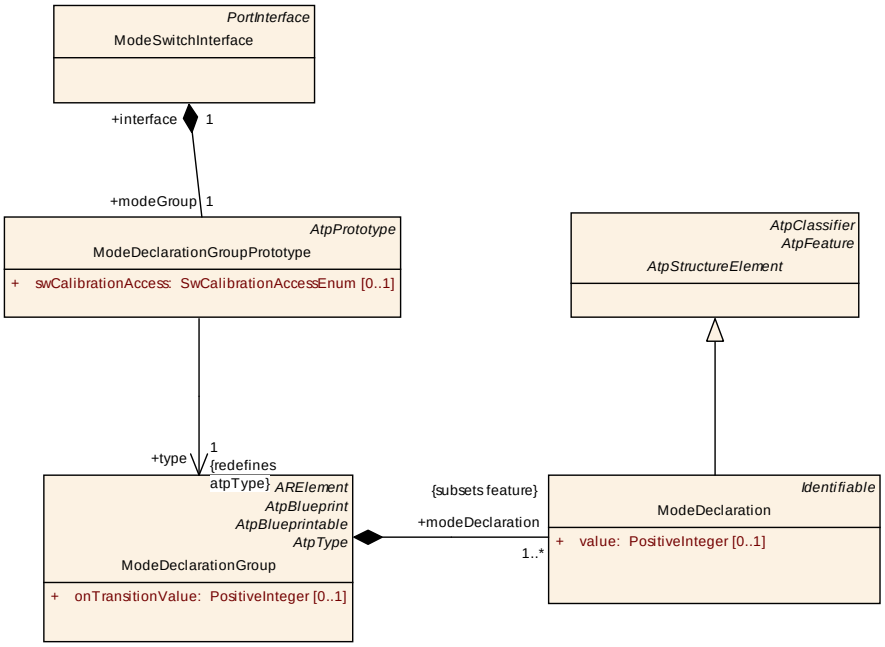

Figure 5.6: Concrete structure element

Class	**AtpClassifier (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure			
Note	A classifier classifies M0 instances according to their features. Or: a classifier is something that has instances - an M1 classifier has M0 instances.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
atpFeature	AtpFeature	*	aggr	This is a feature of the classifier. **Stereotypes:** atpDerived

Table 5.1: AtpClassifier

Class	**AtpFeature (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure			
Note	Features are properties via which a classifier classifies instances. Or: a classifier has features and every M0 instance of it will have those features.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 5.2: AtpFeature

Class	**AtpInstanceRef (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure			
Note	An M0 instance of a classifier may be represented as a tree rooted at that instance, where under each node come the sub-trees representing the instances which act as features under that node. An instance ref specifies a navigation path from any M0 tree-instance of the base (which is a classifier) to a leaf (which is an instance of the target).			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
atpBase	AtpClassifier	1	ref	This is the base from which the navigaion path starts. **Stereotypes:** atpAbstract; atpDerived
atpContext Element (ordered)	AtpPrototype	*	ref	This is one particular step in the navigation path. **Stereotypes:** atpAbstract
atpTarget	AtpFeature	1	ref	This is the target of the instance ref. In other words it is the terminal of the navigation path. **Stereotypes:** atpAbstract

Table 5.3: AtpInstanceRef

Class	**AtpPrototype (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure			
Note	A prototype is a typed feature. A prototype in a classifier indicates that instances of that classifier will have a feature, and the structure of that feature is given by the its type. An instance of that type will play the role indicated by the feature in the owning classifier. A feature is not an instance but an indication of an instance-to-be.			
Base	ARObject,AtpFeature,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
atpType	AtpType	1	ref	This is the type of the feature. **Stereotypes:** atpAbstract

Table 5.4: AtpPrototype

Class	**AtpStructureElement (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure			
Note	A structure element is both a classifier and a feature. As a feature, its structure is given by the feature it owns as a classifier.			
Base	ARObject,AtpClassifier,AtpFeature,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 5.5: AtpStructureElement

Class	**AtpType (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure			
Note	A type is a classifier that may serve to type prototypes. It is a reusable classifier.			
Base	ARObject,AtpClassifier,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 5.6: AtpType

5.1.3 Instance Refs

Instance refs are M1 elements which define a particular navigation within future M0 instance trees of M1 classifiers.

Figure 5.7 shows an M1 instance ref called ”ctrlInRightControllerInPowerWindowType”. In each M0 instance of ”PowerWindowType”, which has the structure shown in Fig. 5.3, this instance ref identifies the bottom most instance on the right side of that figure. It has the composition type ”PowerWindowType” as *base*, the component prototype ”rightController” as *context*, and the port ”ctrl” as *target*. The context servers to navigate the instance tree in a particular path, the right path in this example.

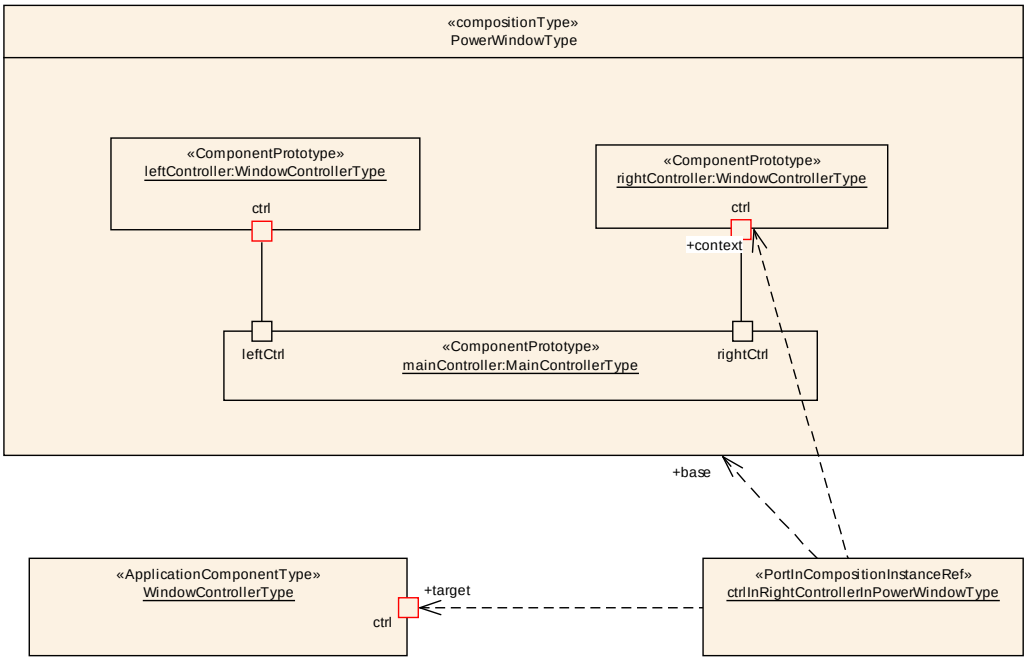

Figure 5.7: M1 Instance Ref

An M1 instance ref is an instance of a concrete subclass of the meta-class `AtpInstanceRef`. Figure 5.8 shows the abstract layer for instance refs. Each instance ref is defined with respect to a *base* which is a classifier. What the instance ref does is specify a particular navigation leading from the root of any M0 instance of the base to an inner instance in the tree. One instance ref, i.e. one navigation, works for *any*

M0 instance of the base. The navigation is specified via a series of *context elements*, which are features, and a *target* which is also a feature.

The ordered set of context features plus the target constitutes the *path* leading from the root to the specified inner instance. In other words, the context starts with the first element of the InstanceRef. The target is always the last element in the InstanceRef-class.

[constr_2530] `InstanceRefs` must be consistent ⌈ The first `atpContextElement` in the path must be an `atpFeature` of the `atpBase`. For all subsequent `atpContextElement`s, they must be an `atpFeature` of the `atpType` of the previous element (which is an `AtpPrototype`). ⌋

[constr_2531] `AtpInstanceRef` must be close to the base ⌈ An `AtpInstanceRef` must be aggregated such that its relationship to the `AtpClassifier` referenced in the role `atpBase` is unambiguous. This is the case in one of the following situations:

- The `AtpInstanceRef` is aggregated within the `AtpFeature` referenced in the role `atpBase`.
- The `atpBase` is the root of the instance tree. It is the `AtpStructureElement` which is not aggregated in another `AtpClassifier`. This in particular the meta-class `System`.

⌋

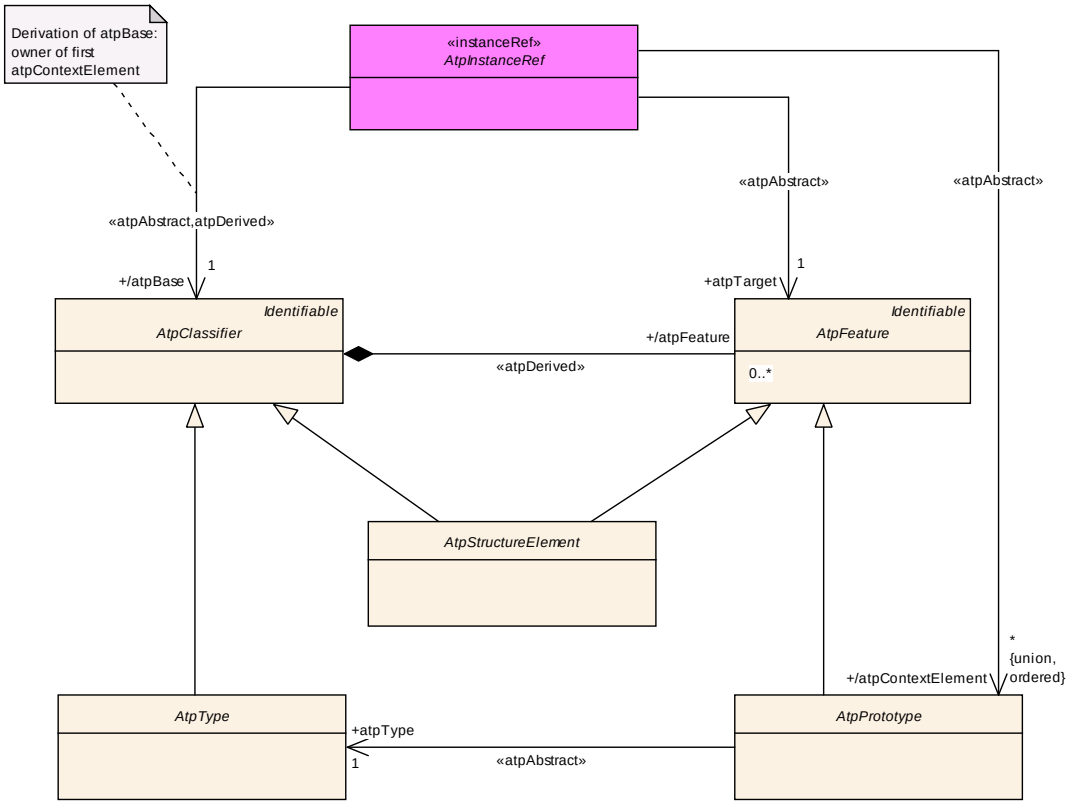

Figure 5.8: Abstract instance Refs

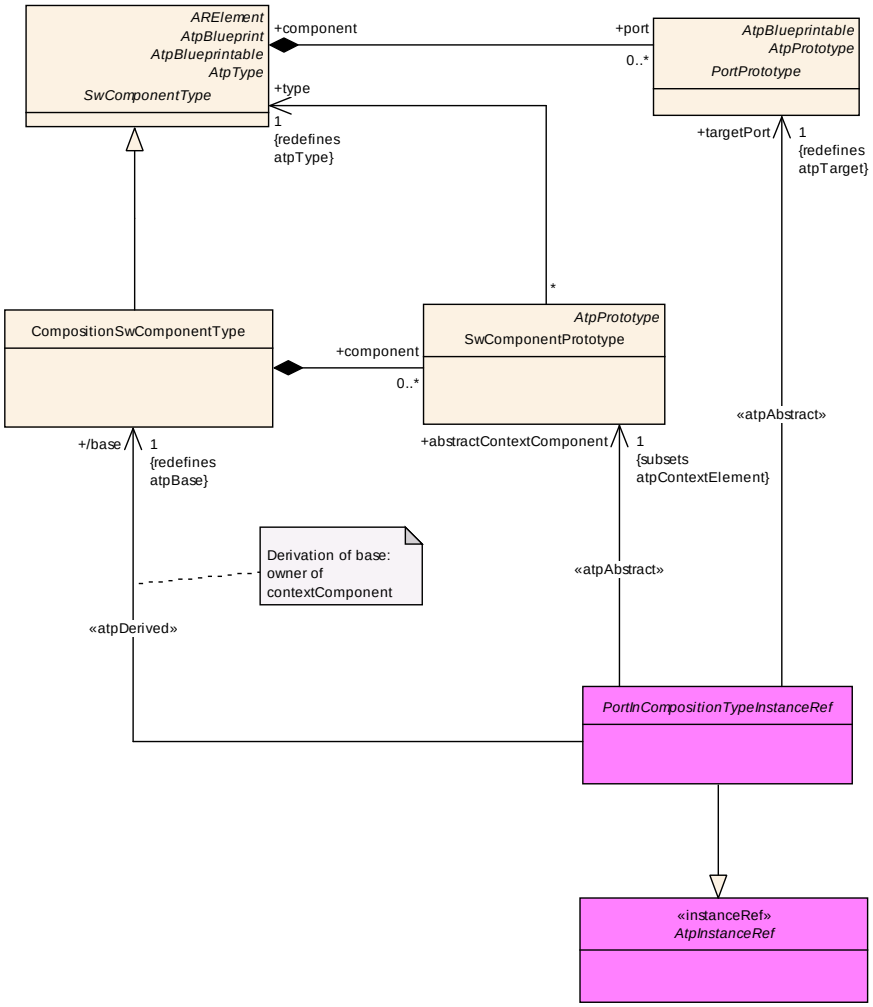

Figure 5.9: Concrete instance ref meta-class

Fig. 5.9 shows a concrete subclass of `AtpInstanceRef`, named ”PortInCompositionInstanceRef”, used to define instance refs which navigate to a port within a composition type via an inner component prototype. This concrete M2 meta-class specifies that the base must be a composition type, that there must be exactly one context element which is a component prototype, and that the target is a port. The instance ref ”ctrlInRightControllerInPowerWindowType” is an instance of ”PortInCompositionInstanceRef”.

Figure 5.10 illustrates how such an instance ref is applied in the meta model:

[TPS_GST_0043] Application of Instance Ref ⌈ Instance refs are applied in the meta model by two representations which shall exist together:

- A **dependency** with stereotype ≪`instanceRef`≫ which represents the intention and contributes to documentation and classtables.
- A corresponding **aggregation** of the concrete subclass of `AtpInstanceRef` in the source of the reference which represents the implementation and therefore contributes to the xml schema.

Note that both representations are relevant and shall exist in the meta model. ⌋

[TPS_GST_0044] Identification of corresponding Instance Ref representations ⌈ The target role name of the **dependency** and the target role name of the corresponding **aggregation** used to model an instance ref (as described in [TPS_GST_0044]) shall be identical.⌋[1]

Please find an example of the application of [TPS_GST_0044] in Figure 5.10. The target role name of the dependency from `DelegationSwConnector` to `PortPrototype` (i.e. `innerPort`) is identical to the target role name of the aggregation of `PortInCompositionTypeInstanceRef` at `DelegationSwConnector`.

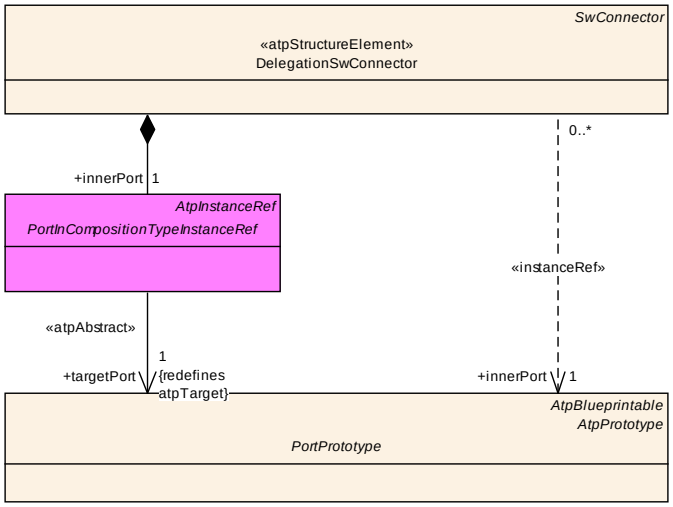

Figure 5.10: Application of an instance Ref

[1]Note that the UML-tool used to generate the diagrams allows to specify role names on dependencies, even if it is not supported by UML.

6 Metamodeling Patterns and Model Transformation

A metamodeling pattern is a parameterized structure which, when applied to actual parameters, yields a regular, non-parameterized structure. A *structure* is just a collection of meta-classes related by associations and aggregations. The benefit of patterns is that they allow recurring structures to be used over and over again without the need to repeat their definitions. This chapter describes the concept of metamodeling patterns as well as their use and notation in the AUTOSAR Metamodel. Another advantage is that the original structure of the metamodel is preserved and not blurred with implementation details.

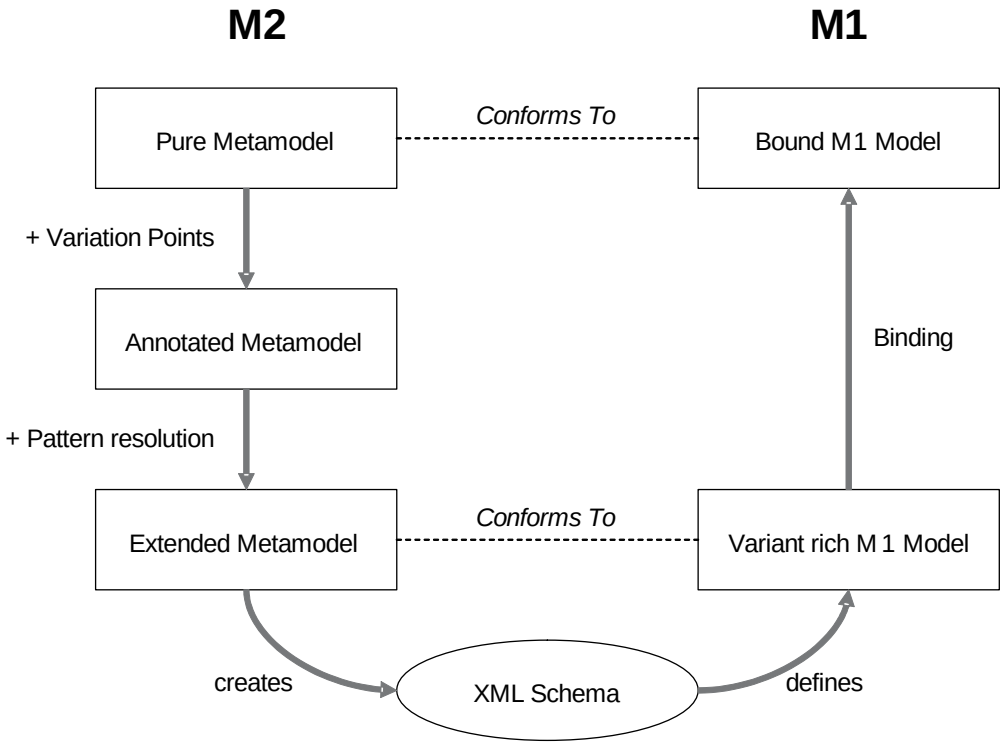

Figure 6.1: Metamodeling Patterns and Model Transformation Overview

Figure 6.1 presents a quick overview of the approach. Thereby it uses as an example the variant handling in AUTOSAR[1]:

1. **(Pure meta model)** AUTOSAR primarily defines an meta model without support for variation, here called *pure meta model*.
2. **(Annotated meta model)** This pure meta model is then annotated with variation points, which is called the *annotated meta model*.

   This annotation consists of pattern specific stereotypes and UML attributes (e.g (latest) binding time).

[1]Note that “model” in this context is a conceptual entity regardless of its representation. In that sense, XML, C and PDF etc. are considered as representations of the “model”. The term “model” relates to the AUTOSAR related content, not physical entities.

Note that finally only the annotated meta model is maintained manually and part of AUTOSAR deliverables.

3. **(Extended meta model)** A **model transformation** converts the *annotated meta model* into the *extended meta model*, which is then used to generate the schema.

   The *extended meta model* differs from the *annotated meta model* in that it adds several more elements that provide all the information which is necessary to fully describe a variation point. It also introduces additional constraints to support the variation points.

   These additional elements are generated by applying patterns to those locations in the *annotated meta model* that are annotated as variation points.

To illustrate consider the pattern `VHUnboundedAggregationPattern` in Fig. 6.2. It shows a parameterized structure consisting of four classes and some aggregations between them. The parameters, shown in the figure between curly brackets ({}) are:

- `WholeClass`, which is a class.
- `PartClass`, which an aggregated class.
- `partRole`, which is the role of the aggregated class.
- `Vh.latestBindingTime`, which is a value of the enumeration type `BindingTimeEnum`.

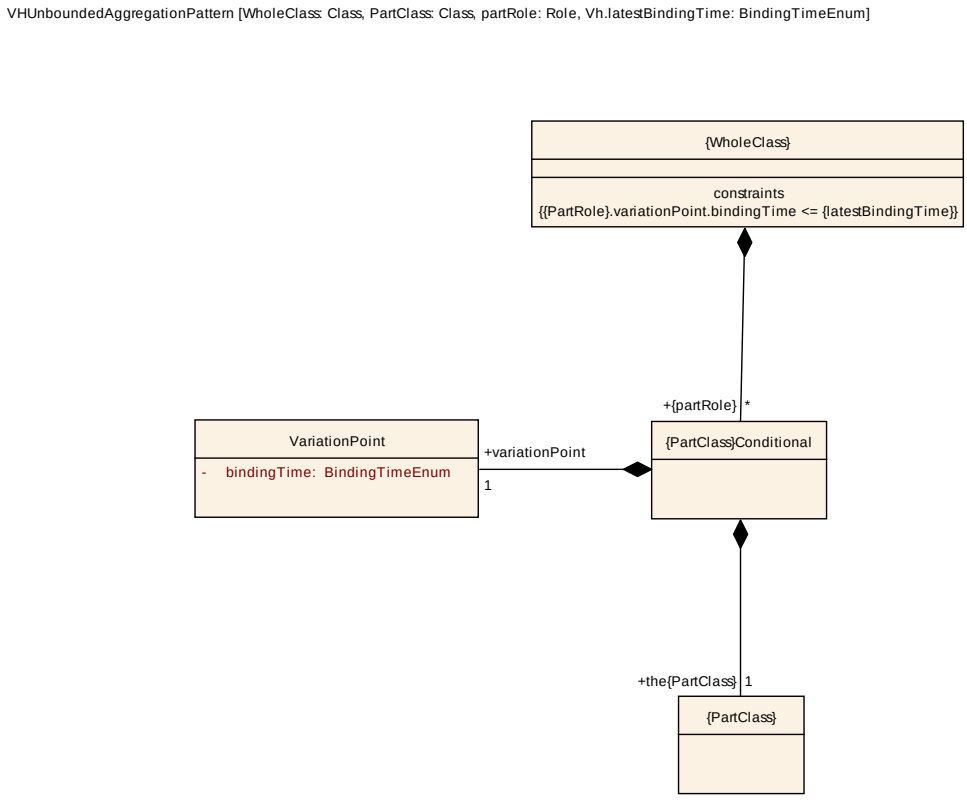

Figure 6.2: Pattern Example (VHUnboundedAggregationPattern)

This pattern may now be applied to actual parameters to yield a non-parameterized structure. For example, by assigning the actual parameters

- `WholeClass` = `ComponentType`
- `PartClass` = `PortPrototype`
- `partRole` = `port`
- `Vh.latestBindingTime` = `SystemDesignTime`

we obtain the structure in Fig. 6.3. Many different structures may be obtained by applying the same pattern to different parameters.

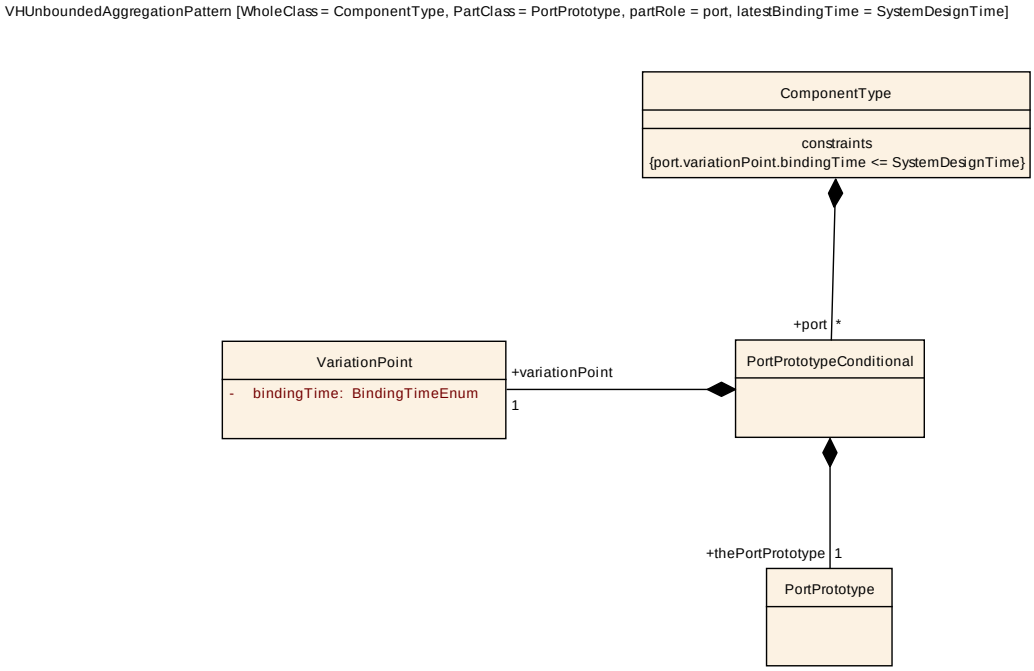

Figure 6.3: Pattern Application Result Example

6.1 Notation for Pattern Application

A recurring structure which has been abstracted into a pattern usually has a well-defined semantics explained in terms of the roles played by the parameters. This semantics will usually suggest an intuitive notation for the application of the pattern, which is of course specific to the pattern at hand.

Figure 6.4 shows a notation for the application of the `VHUnboundedAggregationPattern` pattern on component types and ports. The notation uses an aggregation arrow between the classes playing the role of *WholeClass* and *PartClass*, decorated with the ≪`atpVariation`≫ stereotype. This notation suggests that the class `ComponentType` "semantically aggregates" the class `portPrototype` in role `port` while allowing for variations. This diagram is a notation for a pattern application which results in the diagram in Fig. 6.3

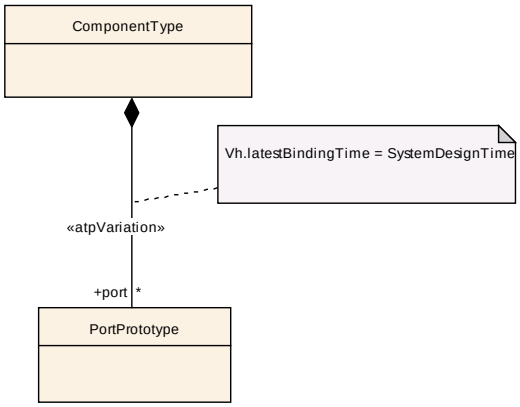

Figure 6.4: Pattern Application Notation Example

A similar notation is used for all variation-related patterns. The particular patterns are described below along with their notations. A major benefit of this kind of notation is that it is similar to a variation-free aggregation. This makes it possible to specify the variations "on top" of the regular, variation-free design instead of meddling with the conceptual design itself. It is however important to keep in mind that aggregation arrows decorated by ≪atpVariation≫ such as the one in Fig. 6.4 are not real aggregations: they are a notation for the application of a pattern (the result of which includes real aggregations).

6.2 Pattern Specification

The specification of a pattern includes:

- The name of the pattern
- The list of parameters
- The parameterized structure
- The notation for the pattern application

The above elements are all specified via a single diagram as illustrated in Fig. 6.5 for the unbounded aggregation pattern.

VHUnboundedAggregationPattern [WholeClass: Class, PartClass: Class, partRole: Role, Vh.latestBindingTime: BindingTimeEnum]

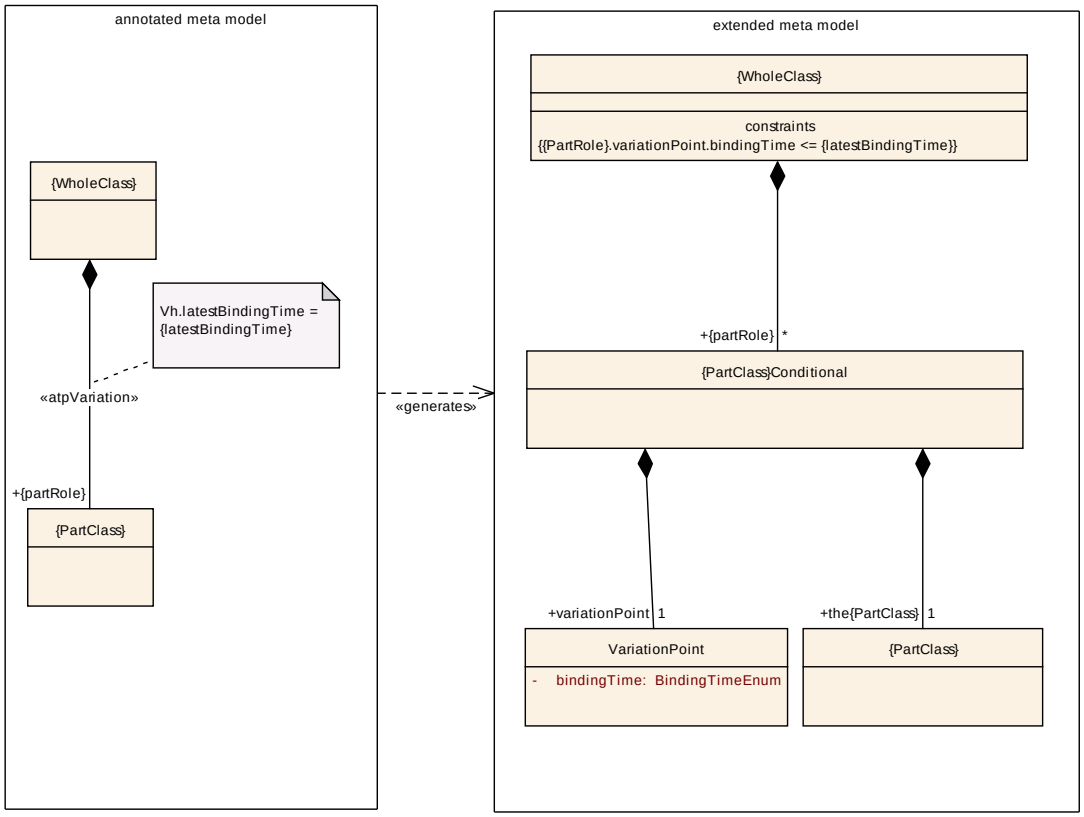

Figure 6.5: Pattern Specification Example

6.3 Model Transformations applied in the Meta-Model

The model transformation pattern mechanism is applied in the MetaModel for

- Variant Handling (chapter 7)
- Referencing (chapter 6.3.2)

6.3.1 Implementing ≪primitive≫s

This section illustrates the implementation of primitives even if it is not yet (R4.0) implemented as a true model transformation[2]. As shown in Figure 6.6, a ≪primitive≫ meta-class is converted to:

1. another primitive with the same name enhanced by "_simple". For xml, this primitive results in a simple type in the generated schema and specifies the implementation details, such as patterns and facets. This meta-class is used for attributes

[2]This approach is implemented in the schema generator.

tagged with `xml.attribute=true` but also as primitive type for attributes tagged with `xml.roleElement=false`, `xml.roleWrapperElement=false`, `xml.typeElement=false`, `xml.typeWrapperElement=false`.

2. a meta-class with the same name. As all other meta-classes, this meta-class inherits from ≪`atpObject`≫ (see chapter 6.3.3). Thus UML-attributes in the meta-model are finally implemented as aggregation of this meta-class unless tagged with `xml.attribute=true`.

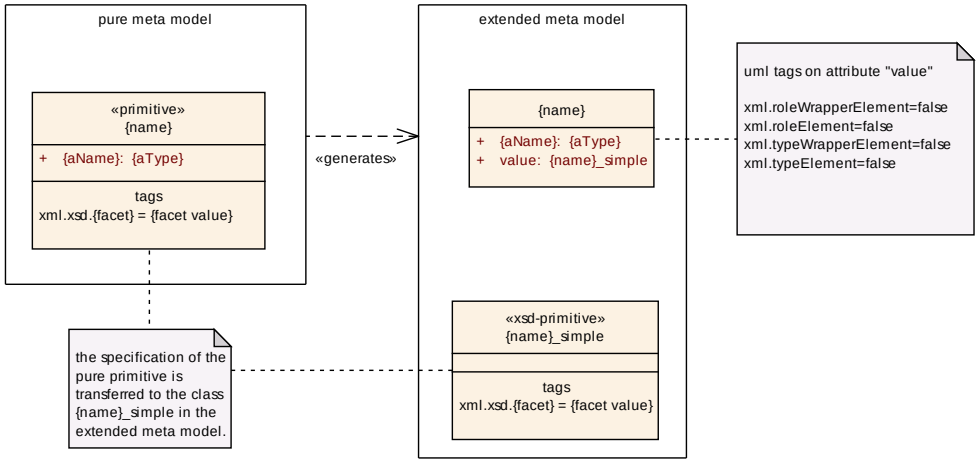

Figure 6.6: Pattern to implement Primitives

6.3.2 Implementing Associations as References

This section illustrates the implementation of associations even if it is not yet (R4.0) implemented as a true model transformation.

[TPS_GST_0020] Establising References ⌈ References between meta-classes are represented as associations. Referenced meta-classes are derived from `Referrable`, mostly being `Identifiable`. Thereby they define a `ShortName` which must be unique within its name space (see [constr_2508])[3]. Therefore references (associations in the meta model) are expressed by

1. by specifying a **case-sensitive** path of `shortName`s (absolute or relative)
2. the destination type of the reference
3. the **case-sensitive** name of the reference base in case of a relative reference

⌋

Note that The term “case-sensitive” indicates that the characters in the sets

`{a b c d e f g h i j k l m n o p q r s t u v w x y z}`

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

{A B C D E F G H I J K L M N O P Q R S T U V W X Y Z}

are respectively considered to be different. In other words case-sensitive paths /X/Y and /x/y are **not** the same.

Note that for association stereotyped with ≪isOfType≫ the role of the reference class is theRoleTref.

Figure 6.7 illustrates the equivalent pattern. An association is converted to an aggregation of an anonymous[4] meta-class with the properties mentioned above. This meta-class inherits from Ref after the transformation according to chapter 6.3.1 was performed.

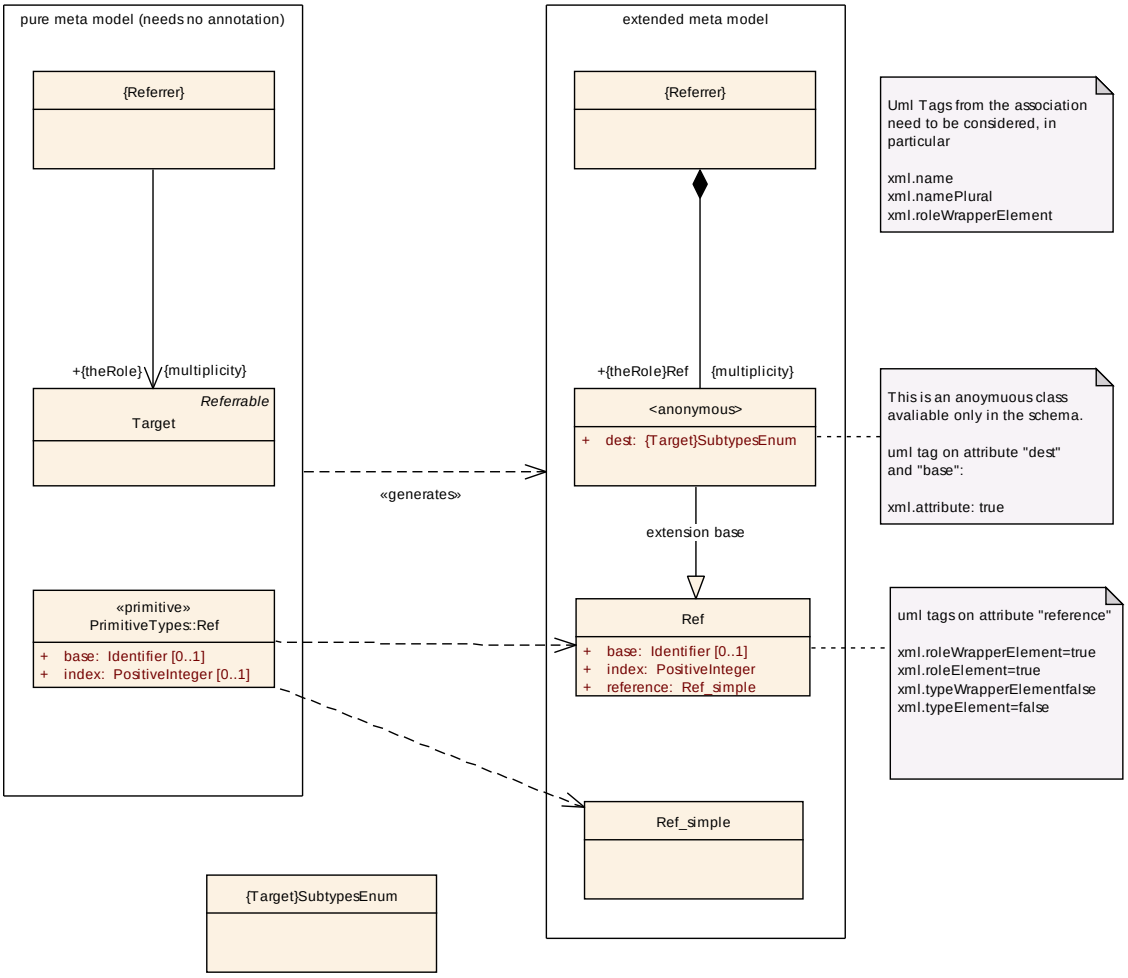

Figure 6.7: Pattern to implement references

[4]This shortcut was possible since the described approach is not implemented as a model transformation but in the schema generator. If a model transformation were applied, a particular meta-class would need to be created for each particular reference.

6.3.2.1 Absolute ShortName-path

ShortName paths are composed of sequences of ShortNames separated by '/'. The following rules apply to ShortName paths used in AUTOSAR:

1. An absolute path is calculated by collecting the shortName of the model elements on the containment path from root element of the model to the referenced element.
2. Absolute paths begin with the character '/'.
3. attribute base is ignored in case of an absolute reference path. But an absolute path can be converted to a relative by removing the ShortName-path specified by the base. It is an error if a base is denoted which does not match to the absolute path.

6.3.2.2 Relative ShortName-path

A relative reference path does not start with the character '/'. A relative reference path can be converted to an absolute path by adding the appropriate ShortName-path of the ReferenceBase in front of the relative ShortName-path.

The appropriate ReferenceBase is identified by

- the attribute base. This denotes the first containing ARPackage visible from the reference which has a referenceBase with shortLabel equal to the base. In other words: as packages are nested the appropriate referenceBase is searched bottom up.

  [constr_2511] Named reference bases must be available ⌈ One of the packages containing a relative reference must have a referenceBase with a shortLabel equal to the base of the reference. ⌋

- alternatively the innermost package which has a referenceBase with default set to "true".

Note that referenceBase is ≪atpSplitable≫. Therefore it is necessary to search for the appropriate ReferenceBase in the entire model. In other words the ReferenceBase shall be searched in all partial models.

Listing 6.1 illustrates the most simple form of relative references. In this case, relative references within a package shall be independent of the package name. Please note the callouts specified as xml comments <!- 1 ->:

1. <!- 1 ->: this a base of relative references. Note that isDefault denotes the fact that it is the default.
2. <!- 2 ->: this is a relative reference. Since the base attribute is not specified, the reference is resolved using the default base specified in <!- 1 ->.

Listing 6.1: Relative Reference with default base

```
<?xml version="1.0"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0 AUTOSAR_4-0-2.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>MyComponent</SHORT-NAME>
      <REFERENCE-BASES>
        <!-- 1 -->
        <REFERENCE-BASE>
          <SHORT-LABEL>default</SHORT-LABEL>
          <IS-DEFAULT>true</IS-DEFAULT>
          <IS-GLOBAL>false</IS-GLOBAL>
          <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
          <PACKAGE-REF DEST="AR-PACKAGE">/MyComponent</PACKAGE-REF>
        </REFERENCE-BASE>
      </REFERENCE-BASES>
      <ELEMENTS>
        <SENDER-RECEIVER-INTERFACE>
          <SHORT-NAME>MyInterface</SHORT-NAME>
          <DATA-ELEMENTS>
            <VARIABLE-DATA-PROTOTYPE>
              <SHORT-NAME>MyData</SHORT-NAME>
              <CATEGORY>VALUE</CATEGORY>
            </VARIABLE-DATA-PROTOTYPE>
          </DATA-ELEMENTS>
        </SENDER-RECEIVER-INTERFACE>
        <APPLICATION-SW-COMPONENT-TYPE>
          <SHORT-NAME>MyComponent</SHORT-NAME>
          <PORTS>
            <P-PORT-PROTOTYPE>
              <SHORT-NAME>MyPort</SHORT-NAME>
              <!-- 2 -->
              <!-- /MyComponent/MyInterface -->
              <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
                 MyInterface</PROVIDED-INTERFACE-TREF>
            </P-PORT-PROTOTYPE>
          </PORTS>
        </APPLICATION-SW-COMPONENT-TYPE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

Listing 6.2 illustrates that there can be multiple reference bases. In this case, in addition to the previous example another base is given for CompuMethods, since those live in a separate package.

1. `<!- 1.1 ->`: this another base of relative references. Note that `isDefault` attribute is missing. It is the same as if it is specified as `false`.

2. `<!- 2.1 ->`: this is a relative reference. The attribute `base` denotes that the base with `shortLabel compum` (defined in `<!- 1.1 ->` shall be used fore the

relative reference. The corresponding absolute reference is /CompuMethods/MyCompu.

Listing 6.2: Relative Reference with explicit base

```
<?xml version="1.0"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema/r4.0 AUTOSAR_4-0-2.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>MyComponent</SHORT-NAME>
      <REFERENCE-BASES>
        <!-- 1 -->
        <REFERENCE-BASE>
          <SHORT-LABEL>default</SHORT-LABEL>
          <IS-DEFAULT>true</IS-DEFAULT>
          <IS-GLOBAL>false</IS-GLOBAL>
          <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
          <PACKAGE-REF DEST="AR-PACKAGE">/MyComponent</PACKAGE-REF>
        </REFERENCE-BASE>
        <REFERENCE-BASE>
          <SHORT-LABEL>compum</SHORT-LABEL>
          <IS-DEFAULT>false</IS-DEFAULT>
          <IS-GLOBAL>false</IS-GLOBAL>
          <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
          <!-- 1.1 -->
          <PACKAGE-REF DEST="AR-PACKAGE">/CompuMethods</PACKAGE-REF>
        </REFERENCE-BASE>
      </REFERENCE-BASES>
      <ELEMENTS>
        <SENDER-RECEIVER-INTERFACE>
          <SHORT-NAME>MyInterface</SHORT-NAME>
          <DATA-ELEMENTS>
            <VARIABLE-DATA-PROTOTYPE>
              <SHORT-NAME>MyData</SHORT-NAME>
              <CATEGORY>VALUE</CATEGORY>
              <SW-DATA-DEF-PROPS>
                <SW-DATA-DEF-PROPS-VARIANTS>
                  <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <!-- 2.1 -->
                    <!-- /CompuMethods/MyCompu -->
                    <COMPU-METHOD-REF DEST="COMPU-METHOD" BASE="compum">
                      MyCompu</COMPU-METHOD-REF>
                  </SW-DATA-DEF-PROPS-CONDITIONAL>
                </SW-DATA-DEF-PROPS-VARIANTS>
              </SW-DATA-DEF-PROPS>
            </VARIABLE-DATA-PROTOTYPE>
          </DATA-ELEMENTS>
        </SENDER-RECEIVER-INTERFACE>
        <APPLICATION-SW-COMPONENT-TYPE>
          <SHORT-NAME>MyComponent</SHORT-NAME>
          <PORTS>
            <P-PORT-PROTOTYPE>
              <SHORT-NAME>MyPort</SHORT-NAME>
              <!-- 2 -->
```

```
            <!-- /MyComponent/MyInterface -->
            <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
              MyInterface</PROVIDED-INTERFACE-TREF>
          </P-PORT-PROTOTYPE>
        </PORTS>
      </APPLICATION-SW-COMPONENT-TYPE>
    </ELEMENTS>
  </AR-PACKAGE>
  <AR-PACKAGE>
    <SHORT-NAME>CompuMethods</SHORT-NAME>
    <ELEMENTS>
      <COMPU-METHOD>
        <SHORT-NAME>MyCompu</SHORT-NAME>
        <CATEGORY>RATFUNC</CATEGORY>
      </COMPU-METHOD>
    </ELEMENTS>
  </AR-PACKAGE>
 </AR-PACKAGES>
</AUTOSAR>
```

Listing 6.3 illustrates that reference bases can be nested. In this case, the base for the CompuMethods is defined relative to the default base. This allows to maintain the entire package relationships at the beginning of a package.

It further on shows that the reference base needs to be search from inside out.

1. <!- 1.1.1 ->: this another base of relative references. Note that base attribute is now explicitly specified. It would also have been possible to relate to the default mechanism here.
2. <!- 1.2 ->: This is the base for units. Note that this base is also defined relative to the default.
3. <!- 2.2 ->: this is a relative reference. The attribute base denotes that the base with shortLabel compum (defined in <!- 1.1.1 -> shall be used fore the relative reference. The corresponding absolute reference is /MyComponent/CompuMethods/MyCompu.
4. <!- 3 ->: This is the package with the CompuMethods. it now lives as nested package in /MyComponent
5. <!- 3.1 ->: This is a reference to a Unit which is relative to the base named units which is defined in <!- 1.2 ->. Note that the reference lives in an inner package which has no package bases.

   This illustrates, that the relevant reference base is in one of the outer packages, but not necessarily in the directly containing package.
6. <!- 4 ->: This is the package with the Units. It lives in a sub package of /MyComponent.

Listing 6.3: Relative Reference with multiple and nested bases

```
<?xml version="1.0"?>
```

```
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_4-0-2.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>MyComponent</SHORT-NAME>
      <REFERENCE-BASES>
        <!--  1 -->
        <REFERENCE-BASE>
          <SHORT-LABEL>default</SHORT-LABEL>
          <IS-DEFAULT>true</IS-DEFAULT>
          <IS-GLOBAL>false</IS-GLOBAL>
          <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
          <PACKAGE-REF DEST="AR-PACKAGE">/MyComponent</PACKAGE-REF>
        </REFERENCE-BASE>
        <!-- 1.1 -->
        <REFERENCE-BASE>
          <SHORT-LABEL>compum</SHORT-LABEL>
          <IS-DEFAULT>false</IS-DEFAULT>
          <IS-GLOBAL>false</IS-GLOBAL>
          <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
          <!-- 1.1.1 -->
          <PACKAGE-REF BASE="default" DEST="AR-PACKAGE">NestedCompuMethods<
              /PACKAGE-REF>
        </REFERENCE-BASE>
        <REFERENCE-BASE>
          <SHORT-LABEL>unit</SHORT-LABEL>
          <IS-DEFAULT>false</IS-DEFAULT>
          <IS-GLOBAL>false</IS-GLOBAL>
          <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
          <!-- 1.2 -->
          <PACKAGE-REF DEST="AR-PACKAGE">/MyUnits</PACKAGE-REF>
        </REFERENCE-BASE>
      </REFERENCE-BASES>
      <ELEMENTS>
        <SENDER-RECEIVER-INTERFACE>
          <SHORT-NAME>MyInterface</SHORT-NAME>
          <DATA-ELEMENTS>
            <VARIABLE-DATA-PROTOTYPE>
              <SHORT-NAME>MyData</SHORT-NAME>
              <CATEGORY>VALUE</CATEGORY>
              <SW-DATA-DEF-PROPS>
                <SW-DATA-DEF-PROPS-VARIANTS>
                  <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <!-- 2.2 -->
                    <!-- /MyComponent/NestedCompuMethods/MyCompu  -->
                    <COMPU-METHOD-REF DEST="COMPU-METHOD" BASE="compum">
                      MyCompu</COMPU-METHOD-REF>
                  </SW-DATA-DEF-PROPS-CONDITIONAL>
                </SW-DATA-DEF-PROPS-VARIANTS>
              </SW-DATA-DEF-PROPS>
            </VARIABLE-DATA-PROTOTYPE>
          </DATA-ELEMENTS>
        </SENDER-RECEIVER-INTERFACE>
        <APPLICATION-SW-COMPONENT-TYPE>
          <SHORT-NAME>MyComponent</SHORT-NAME>
```

```
          <PORTS>
            <P-PORT-PROTOTYPE>
              <SHORT-NAME>MyPort</SHORT-NAME>
              <!-- 2 -->
              <!-- /MyComponent/MyInterface -->
              <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
                 MyInterface</PROVIDED-INTERFACE-TREF>
            </P-PORT-PROTOTYPE>
          </PORTS>
        </APPLICATION-SW-COMPONENT-TYPE>
      </ELEMENTS>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <!-- 3 -->
          <SHORT-NAME>NestedCompuMethods</SHORT-NAME>
          <ELEMENTS>
            <COMPU-METHOD>
              <SHORT-NAME>MyCompu</SHORT-NAME>
              <CATEGORY>RATFUNC</CATEGORY>
              <!-- 3.1 -->
              <!-- /MyUnits/kmh -->
              <UNIT-REF BASE="unit" DEST="UNIT">kmh</UNIT-REF>
            </COMPU-METHOD>
          </ELEMENTS>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
    <!-- 4 -->
    <AR-PACKAGE>
      <SHORT-NAME>MyUnits</SHORT-NAME>
      <ELEMENTS>
        <UNIT>
          <SHORT-NAME>kmh</SHORT-NAME>
        </UNIT>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

Listing 6.4 illustrates a relative reference with a global target.

1. `<!- 1 ->` denotes the `referenceBase`. Thereby it declares that via this base `Chapter` and `TextualConstraint`s are globally unique.
2. `<!- 2 ->` "ChR_4711" is a globally referrable target
3. `<!- 3 ->` refers to a textual constraint ("Constr_0815") via the base in `<!- 1 ->`
4. `<!- 5 ->` "Constr_0815" is a globally referrable `TextualConstraint`
5. `<!- 6 ->` refers to a chapter ("ChR_4711") via the base in `<!- 1 ->`

Listing 6.4: Relative Reference with global reference

```
<?xml version="1.0"?>
```

```
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
    //autosar.org/schema/r4.0" xsi:schemaLocation="http://autosar.org/schema
    /r4.0␣AUTOSAR_4-0-2.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Demo</SHORT-NAME>
      <REFERENCE-BASES>
        <!--  1 -->
        <REFERENCE-BASE>
          <SHORT-LABEL>globals</SHORT-LABEL>
          <IS-DEFAULT>false</IS-DEFAULT>
          <IS-GLOBAL>true</IS-GLOBAL>
          <BASE-IS-THIS-PACKAGE>true</BASE-IS-THIS-PACKAGE>
          <GLOBAL-ELEMENTS>
            <GLOBAL-ELEMENT>TRACEABLE</GLOBAL-ELEMENT>
            <GLOBAL-ELEMENT>CHAPTER</GLOBAL-ELEMENT>
          </GLOBAL-ELEMENTS>

        </REFERENCE-BASE>
      </REFERENCE-BASES>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>mySupPackage</SHORT-NAME>
          <ELEMENTS>
            <DOCUMENTATION>
              <SHORT-NAME>myDoc</SHORT-NAME>
              <DOCUMENTATION-CONTENT>
                <!-- 2 -->
                <CHAPTER>
                  <SHORT-NAME>ChR_4711</SHORT-NAME>
                  <LONG-NAME>
                    <L-4 L="EN">More details on Requirement 4711</L-4>
                  </LONG-NAME>
                  <P>
                    <L-1 L="EN">Lorem ipsum dolor sit amet, consetetur
                        sadipscing elitr, sed diam nonumy eirmod tempor
                        invidunt ut labore et dolore magna aliquyam erat,<
                        XREF>
                      <!-- 3 -->
                      <!-- /Demo/mySubPackage/myDoc/Ch_001/Constr_0815
                           global: /Demo@/Const_0815 -->
                      <REFERRABLE-REF BASE="globals" DEST="TRACEABLE">
                          Constr_0815</REFERRABLE-REF></XREF>autem vel eum
                          iriure dolor in hendrerit in vulputate velit esse
                          molestie consequat, vel illum dolore eu</L-1>
                  </P>
                </CHAPTER>
                <CHAPTER>
                  <SHORT-NAME>Ch_001</SHORT-NAME>
                  <LONG-NAME>
                    <L-4 L="EN">an implementation</L-4>
                  </LONG-NAME>
                  <!-- 5 -->
                  <TRACE>
                    <SHORT-NAME>Constr_0815</SHORT-NAME>
                    <LONG-NAME>
```

```
                    <L-4 L="EN">This is my specific constraint</L-4>
                  </LONG-NAME>
                  <P>
                    <L-1 L="EN">Lorem ipsum dolor sit amet, consetetur
                       sadipscing elitr, sed diam nonumy eirmod tempor
                       invidunt<XREF>
                      <!-- 6 -->
                      <!-- /Demo/mySupPackage/myDoc/ChR_4711
                           global: global: /Demo@/ChR_4711 -->
                      <REFERRABLE-REF BASE="globals" DEST="CHAPTER">
                         ChR_4711</REFERRABLE-REF></XREF>autem vel eum
                         iriure dolor in hendrerit in vulputate velit
                         esse molestie consequat, vel illum dolore eu</L
                         -1>
                  </P>
                </TRACE>
              </CHAPTER>
            </DOCUMENTATION-CONTENT>
          </DOCUMENTATION>
        </ELEMENTS>
      </AR-PACKAGE>
    </AR-PACKAGES>
  </AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>
```

6.3.2.3 Destination Type

The destination type (specified as attribute `dest` in the reference elements) defines the type of the referenced object. The destination type can also reference subclasses (abstract and concrete). The value of this attribute is

- If the reference in the meta model points to an abstract class, the value of `dest` is the XML-name of the abstract class or any subclass of the same. Even if `dest` is the XML-name of an abstract class, the target of the reference can be an instance of any concrete class derived from the denoted abstract class.
- If the reference in the meta model points to a concrete class, the value of `dest` is the name of this class. The target of the reference can only be an instance of the denoted class.

The destination type improves the robustness of the XML descriptions such as:

- A tool can find references which refer to objects of the wrong type.
- If the referenced object is not available the, tool can indicate the correct type resp. instantiate a proper proxy.

But if the possible values of `dest` would not include abstract classes, this would cause the problem:

- It requires maintenance of the references if the reference target is changed to another subclass, even if the reference would not care about this.
- It does not propagate information of the meta model to the XML schema

Figure 6.8 illustrates the implementation of an association.

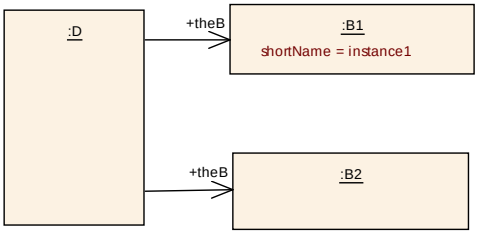

Figure 6.8: Example for linking

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

6.3.3 ≪atpObject≫ARObject

In the Autosar Metamodel there shall be attributes which are applicable to all concrete meta-classes.

Class	**ARObject (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ArObject			
Note	Implicit base class of all classes in meta-model.			
Base				
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
checksum	String	0..1	attr	Checksum calculated from the • attributes • aggregations and aggregated non-identifiables • references **Tags:** xml.attribute=true; xml.name=S
timestamp	DateTime	0..1	attr	The timestamp of the creation or modification of an instance, its attributes, references or aggregated non-identifiables. **Tags:** xml.attribute=true; xml.name=T

Table 6.1: ARObject

This can be considered as a model transformation according to figure 6.9 which applies all meta-classes of ≪`atpObject`≫ (in particular `ARObject`) as superclass to all base classes.

Note that the pattern does not really require arguments.

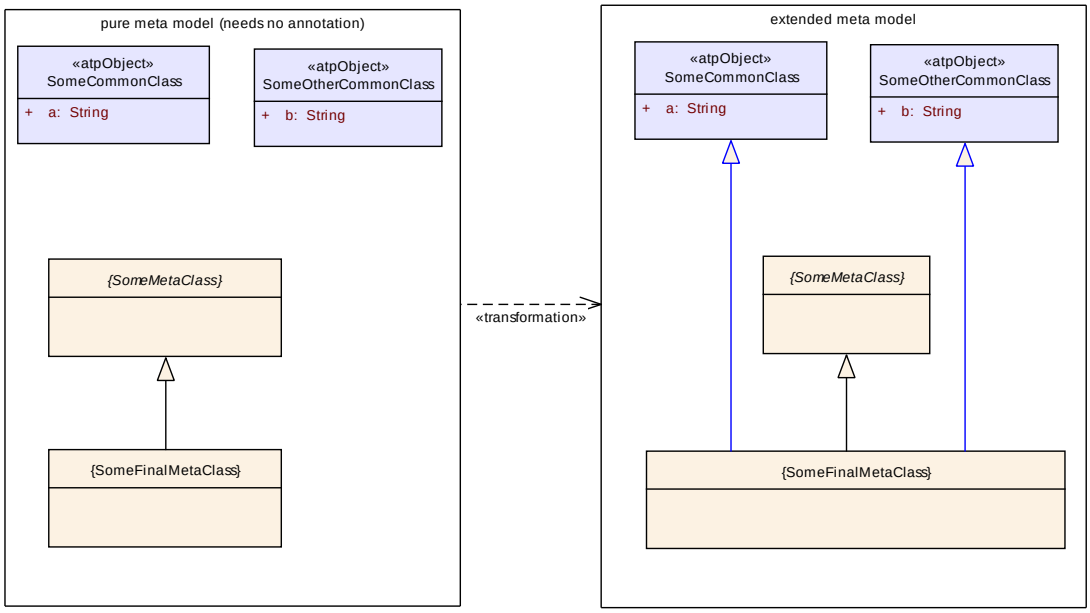

Figure 6.9: Pattern for ≪`atpObject`≫

.

7 Variant Handling

7.1 Introduction

The motivation for *Variant Handling* in AUTOSAR are to build a bridge between OEM's and suppliers, to avoid redundancy between artifacts, and to provide a basis for expressing basic product lines in AUTOSAR.

Of course, variant handling concepts do already exist at most companies, but they are typically not standardized (beyond company borders), and thus it is difficult for OEM's and suppliers to talk to each other on this subject.

Consider the following example. An OEM sends a model which contains variants to a supplier. The supplier generates code from this model, but does not resolve all variants. What the OEM gets back is object code with some variants bound, and other variants left "open" for binding at load time. This can only work if both parties speak the same language, and have the same understanding about variants. And quite often, more than two parties are involved.

Hence, variant handling in AUTOSAR is mostly about *documenting* variants:

- *Variation Points* are locations in the model that are variable. That is, they may not exist in all variants, or may have different characteristics in different variants.
- The *Binding Time* is the latest possible time when a variation point may be bound.
- *Binding Expressions* specify under which condition(s) a variable element exists, or determine certain variable characteristics.

7.1.1 A Quick Overview

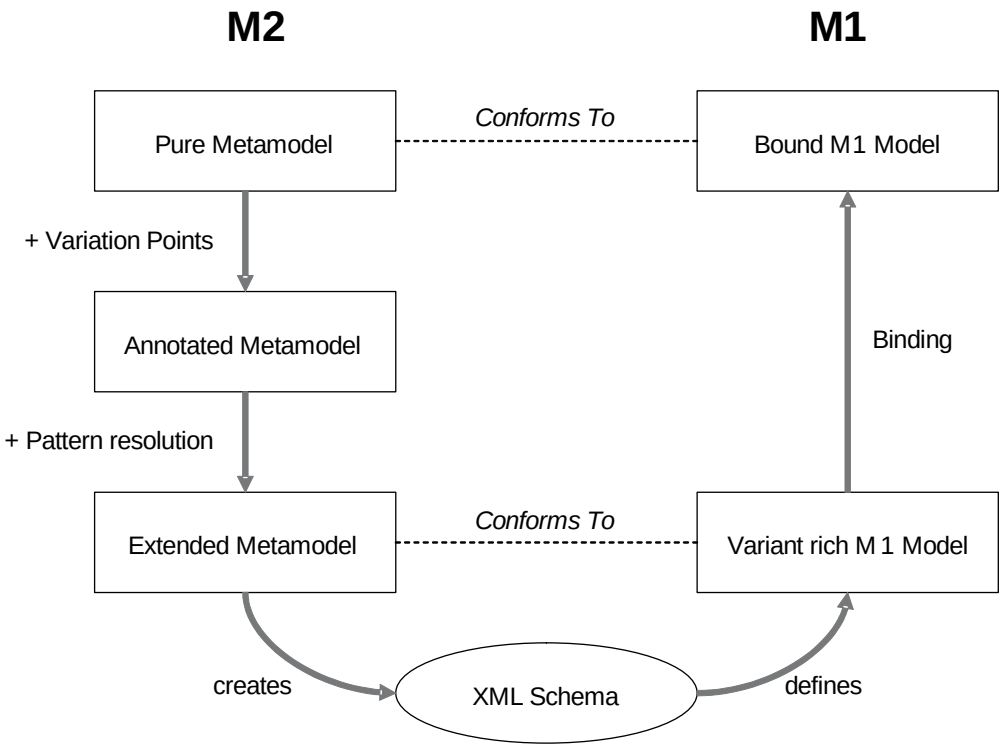

Figure 7.1: Quick Overview

In this section, we continue the meta model transformation overview from Chapter 6.

1. (Recapitulation from Chapter 6.) AUTOSAR starts with the *pure meta model*. The *annotated meta model* has the same structure, but marks those locations that are variation points with a stereotype (≪`atpVariation`≫). In this section, we will define patterns that show to transform those locations into a structure that has all the information that is necessary to implement variation points.
2. All this happens on the M2 model. After the XML schema has been created, the focus shifts to M1. This is where the variants are bound.
3. **(Variant rich M1 model)** A *variant rich M1 model* (which must conform to the rules defined by the *extended meta model*) is transformed into one particular variant by binding the variation points.
4. **(Bound M1 model) [constr_2503] Bound model must be compliant to the meta model** ⌈ The *completely*[1] *bound M1 model* must adhere to the *pure meta model* with respect to consistency rules and semantic constraints defined in the related template specifications. Especially, the multiplicities in the bound model must conform to the multiplicities and the constraints of the *pure meta model*. ⌋

   For example, in an invariant model, ports may have either 1:n or n:1 connections. On the other hand, this rule does not apply to a variant rich model because the

[1] Completely bound includes post build!

variants might overlap each other which results in m:n connections. After binding the variant rich model and extracting one particular variant, the rules of an invariant model apply again.

As another example, the interface compatibility rules can only be applied to a particular variant.

Note that the existence of *PostBuild* variation points implies in practice that the completely bound M1 model not necessarily exists as an artifact. The reason is that the *PostBuild* variation points are bound at startup of the ECU and obviously cannot be resolved in the M1 model. However, the resulting M0 model conforms to the bound M1 model.

If the bound M1 model is saved as artifact, then deselected objects shall be removed. It is up to the process whether the variation points for the selected objects are left in the artifacts. If they are left in, the binding function shall be removed (so that it does not look like an unbound variation point).

7.1.2 Variant Handling and Methodology

As shown before, variant handling takes place on different levels and in different stages in the methodology, and so may have impact on different work products. In fact, every work product can now contain variations. In AUTOSAR, we specify only how variations are represented in ARXML. Everything else is implementation specific and not standardized on M2 level.

But in general, one could say that the methodology without variant handling can be extended to variant handling by introducing resolution of variation as first action of each activity.

For example, a variant rich software component description (represented by an ARXML file) can be preprocessed to a bound software component description by a variant resolving tool, and then handled as defined in the non-variant methodology.

Every variation point starts in the XML code, but is not necessarily resolved in the XML code. If it is not resolved in a processing step, it needs to be transferred to the output artifacts. For example, this means that generated C headers may contain `#ifdef` statements (for *PreCompile Time* variability).

Another example is PostBuild variation, where the generated object code contains conditional statements for implementing the PostBuild variation points, as specified in the XML representation.

7.1.3 How Variant Handling is implemented in the meta-model

Variant Handling in AUTOSAR consists of the following steps[2]:

1. In the *annotated meta model* – that is, on M2 level – all locations which may exhibit variability are annotated as such. This is realized by applying the stereotype ≪atpVariation≫.

   There are four kinds of locations in the meta model which may exhibit variability:

   - Aggregations (see Section 7.2)
   - Associations (see Section 7.3)
   - Attribute Values (see Section 7.4)
   - Classes providing property sets (see Section 7.5)

   Each of the sections referenced above describes a *pattern* that is used to transform the stereotype ≪atpVariation≫ and its associated element into a structure which provides all the information that is necessary to describe a variation point. In other words, they describe the transformation from the *annotated meta model* to the *enhanced meta model*.

   The advantage of this approach is – besides not cluttering the meta model with lots of variant handling related information – that the stereotype ≪atpVariation≫ now also serves as a way to document all sites in the meta model where variations may occur.

   Note: at this point, we are still in the *annotated meta model*. Hence, we do only define which variations are possible, but do not provide the means to specify a condition for this variation (i.e., *when* it occurs), or means to select a particular variant.

2. In all of the aforementioned patterns, an UML tag `Vh.LatestBindingTime`[3] is associated with the stereotype. More precisely, it is attached to the element that has the stereotype ≪atpVariation≫.

   This tag is applied to M2 elements. It does *not* specify the binding time for the variation point, but puts an *upper limit* on the possible values for the binding time of the variation point.

   The binding time of an individual variation point is specified in the attribute `bindingTime` its `ConditionByFormula` or `AttributeValueVariationPoint` element. Hence the name *latest*; the value stated defined with `Vh.LatestBindingTime` is the latest binding time that can be assigned to this element.

[2] In this section, and continuing in the reminder of this chapter, we are heavily borrowing terms from Section 6, especially Figure 6.1

[3] AUTOSAR guidelines require the use of a prefix for tagged values. In our case, the prefix “Vh” stands for “Variant Handling”.

`Vh.LatestBindingTime` is described in more detail in Sections 7.1.6 and 7.6.4; the binding times are explained in Section 7.10.

3. Before the XML schema can be generated from the meta model (for example, by the metamodel tool), all variation points are transformed into a structure that allows to specify detailed information for each variation point, including its actual binding time, and the conditions for when the variation occurs.

   This is done by applying the patterns mentioned in step 1 to the *annotated meta model*. The result of this transformation is the *expanded meta model*.

   In other words, individual variations are defined on M1 level using the means provided on M2 level.

   Details for the individual patterns are described in the respective Sections 7.2, 7.3, 7.4, and 7.5, while the class `VariationPoint` is explained in Section 7.6.

4. The model designer finally specifies the condition(s) and the latest binding time for this variation. This is done in the *variant rich model* on M1 level.

   The condition specifies under what circumstances a particular variation becomes active. For example, it may select one of several alternatives; in this case, the conditions should better be mutually exclusive. Such a condition is, in a nutshell, an expression with system constants as operands.

   See Section 7.6.8 and Section 4.8 earlier in this document for more details on conditions.

5. The next step is to *choose* a particular variant. This may occur at any time before the binding time defined for this variation.

   Conditions are, as already said in the step 4, essentially expressions using system constants as operands. Hence, a particular variant of a system is chosen by assigning values to all system constants which are referenced by all variation points which are relevant for the given binding time.

   It should be noted here that not all variants are chosen at the same time. Instead, there may (and typically will) be several “waves” of binding variations, each at a different time. Similarly, not all system constants need to be determined at the same time – the only restriction here is that their values must be available at the time they are needed.

   In addition, at this stage the model still contains information about possible variants. In the beginning, this are *all* variants. At a later stage, some of those variants are already bound (see Step 6) and the model only contains information on a subset of the original variants.

   This step is described in detail in Section 7.8.

6. When a variation point is bound, the selection in step 5 is implemented and the variation is resolved.

This means that all elements which aggregate a `VariationPoint` whose condition evaluates to *false* are removed from the model, and all `AttributeValueVariationPoint` elements get their `value` attribute fixed.

As indicated in Section 6, not *all* variations are bound at the same (binding) time. At each binding step, unbound variations may still be in existence, and can be bound at a later (binding) time[4]. In other words, Steps 5 and 6 are usually iterated several times.

7.1.4 Not every element in the meta model may be variant

The stereotype ≪`atpVariation`≫ is only allowed for specific elements. These elements are defined by the respective AUTOSAR templates.

The reason for this is that when the stereotype ≪`atpVariation`≫ is attached to an element, there are consequences for other elements. For example, when a port is variable, all the connections from and to this port are also variable. The appropriate measures have to be described precisely in the associated AUTOSAR template. Hence, only a limited number of locations in the meta model support variation.

The list of UML elements that allow variation points can be retrieved by querying the stereotypes ≪`atpVariation`≫ in the meta model.

Technically, this restriction is implemented as follows. The XML schema is equivalent to the *extended meta model*, and simply does not allow for attaching `VariationPoint` or other variant handling related elements to arbitrary locations. The schema makes sure that only those locations that are tagged with `atpVariation` in the *annotated meta model* may contain such elements.

7.1.5 Variation Points are optional, even for variant elements

The stereotype ≪`atpVariation`≫ marks a model element on M2 level as variable. However, the element is not necessarily variable *every time* when it is used on M1 level – on the contrary, variability should only be employed when needed.

This might imply that if a model element is potentially variable, there is a significant overhead on M1 level even if the element is used in a non-variant way. However, this is not the case: the `VariationPoint` element that serves as a starting point for variability information is always optional. Hence, it only needs to be provided only when there is need for it – if absent, there is no variability.

7.1.6 A note on Binding Times

In AUTOSAR, we handle binding times on three different levels:

[4] Of course, there is no binding time after *PostBuild*.

1. There is the binding time that is specified on M0 level. This the value of the attribute `bindingTime` of the element `ConditionByFormula` (see Figure 7.8) and of `AttributeValueVariationPoint` (see Figure 7.4).

   The value of `bindingTime` defines the *latest* binding time for this particular variation point. A variation may be chosen and bound earlier, but not later than `bindingTime`. In consequence, a variation may be bound as soon as all involved system constants have values.

   On the other hand, a system constant must have a value at the "earliest" binding time in which it is required. If we do not have this, then it might be the case that a system constant value is changed during the development cycle which again leads to inconsistencies.

2. There are the individual applications of the stereotype ≪`atpVariation`≫ in the annotated meta model on M2 level. For each such application, a tag `Vh.latestBindingTime` is associated with the stereotype.

   An element that is tagged with the stereotype ≪`atpVariation`≫ is transformed into a more elaborated structure which provides all the information that is necessary to define variation points. This transformation always introduces a `ConditionByFormula` or a `AttributeValueVariationPoint`, i.e., an element that has an attribute `bindingTime`.

   [constr_2504] Constraint to latest binding time ⌈ The tag `Vh.latestBindingTime` *constraints* the value of the attribute `bindingTime` from step 1. Hence, it defines what is allowed as latest binding time for this particular application of ≪`atpVariation`≫. ⌋

3. There are the patterns that describe how the stereotype ≪`atpVariation`≫ is translated into more elaborated UML constructs which provide all the information that is necessary to define variation points.

   These patterns may again restrict the potential range of values for `Vh.latestBindingTime`. These particular restrictions are defined in Sections 7.2.2, 7.3.2, 7.4.4, and 7.5.3.

7.1.7 A note on the impact of Variant Handling on the XML Schema

Naturally, variant handling requires some changes and extensions in the XML schema for M1 AUTOSAR models. In general, those changes are kept as simple as possible. With the exception of the *Property Set Pattern* (Section 7.5), these changes to not introduce significant structural changes in the XML, but rather add and sometimes rename elements.

Pattern specific issues of the respective XML code are documented in the respective subsections of the individual patterns (Sections 7.2.4, 7.3.4, 7.4.6, and 7.5.5).

7.1.8 Patterns are independent of each other

We define four patterns for defining variation points in aggregations, associations, attribute values and property sets. These patterns describe how the ≪atpVariation≫ translates an existing UML construct – usually a relation or an attribute – into one that provides support for describing variants.

These patterns can be combined in any fashion. For example, an element that has variant attributes may be part of a variant aggregation. Or, individual attributes in the property set pattern may be variant themselves.

7.1.9 A note on multiplicites in the Variant Handling Patterns

The Variant Handling patterns need to transform the upper multiplicities for certain aggregations associations within the pattern. Typically, the upper multiplicity of an aggregation is raised to $*$.

This leads to the following constellation:

1. In the *pure meta model*, the multiplicity is $m \ldots n$.
2. In the *annotated meta model*, the multiplicity is still $m \ldots n$.
3. In the *extended meta model*, the multiplicity is increased to $m \ldots *$.

Of these, the *extended meta model* is the one that adds additional structure to describe *all* possible variants. The consequence is that the *variant rich M1 model*, which corresponds to the *extended (M2) meta model*, may aggregate or associate more elements than there would be allowed in the original $m \ldots n$ multiplicity of the *pure meta model*.

In other words, the *variant rich M1 model* violates the multiplicity that is defined in the the *pure meta model*.

However, it is not as serious as it sounds, because the violation only takes place before the binding is completed. The *variant rich M1 model* starts with all possible variants, but as we have described in Section 7.1.3, the “excess” variations are removed during the binding process.

After the binding has been completed – in the *bound M1 model* – the multiplicity must again be within the limit as defined in the *pure meta model*. This is ensured by semantic constraints that are part of the pattern.

7.1.10 A note on the application of the variant handling patterns

The patterns are resolved in the following order:

1. *Property Set Pattern*(Section 7.5)
2. *Aggregation Pattern* (Section 7.2)

3. *Association Pattern* (Section 7.3)
4. *Attribute Value Pattern* (Section 7.4)

It should be noted that this is not the sequence in which the patterns are presented in this document; namely, the *property set pattern* comes last. This is because our descriptions start with those patterns that are easiest to understand, and then proceed to the more complex ones.

7.2 *Aggregation Pattern* for Variation Points

7.2.1 Description

Figure 7.2 illustrates how the metamodel tool transforms an aggregation into an M2 model with variation information.

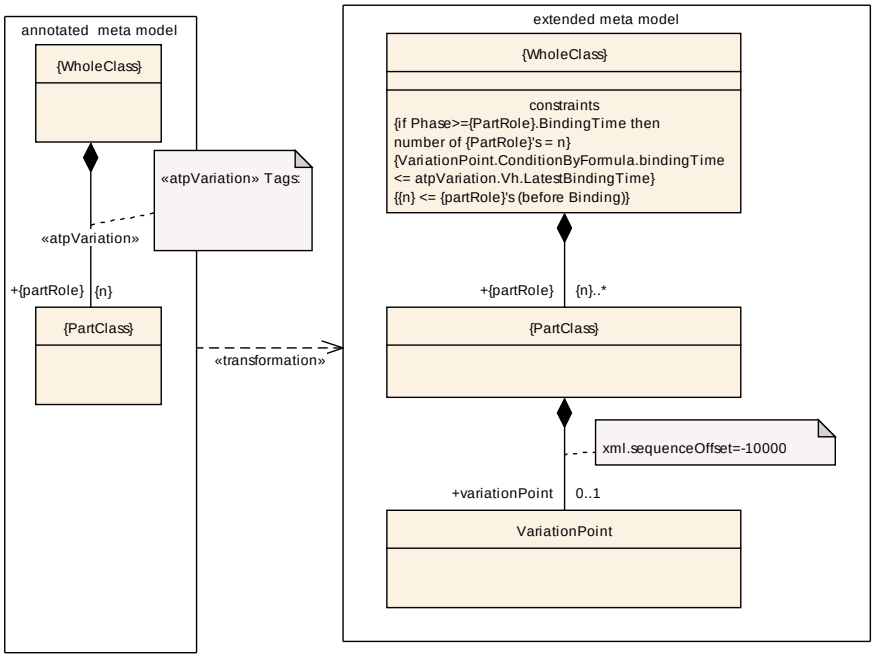

Figure 7.2: Variation Point in Aggregation

On the left side, `{WholeClass}` aggregates `{PartClass}`. This aggregation is subject to variation: the aggregation may or may not exist in the bound model. This is indicated by applying the stereotype ≪`atpVariation`≫ to the association.

Note that the left side only consists of `{WholeClass}`, ≪`atpVariation`≫ and `{PartClass}`, and does not include a condition that determines whether the variation exists. Such a condition is added as part of `VariationPoint` on the right side. In any case, it would not make sense to specify such a condition *here*, because the diagram is on M2 level, while the condition can only be filled in for a concrete variation point, e.g., on M1 level.

In a nutshell, the transformation works as follows:

1. {WholeClass} still aggregates {PartClass} directly.
2. The multiplicity of {partRole} changes from {n} to {n}...*.

   The reason for the change in the multiplicity is that a model that contains variants naturally will contain more {PartClass} elements than a model in which the variants have already been bound[5].

   This is indicated by the constraint

   [constr_2505] Multiplicity after binding ⌈

   if Phase $\geq$ {partRole}.BindingTime *then* number of {partRole}'s = n

   ⌋

   where n is the final number of *PartRole* elements.
3. {PartClass} aggregates a VariationPoint. As described in Section 7.6.1, VariationPoint aggregates further information about this variability, especially the binding time and the condition which guards the variation.

   The multiplicity of VariationPoint is $0 \ldots 1$, meaning that the variation point is optional. If no variation point is given, then the aggregation from {WholeClass} to {PartClass} is invariant. This can be seen as equivalent to a variation point where the condition always evaluates to *true*.

   The reason for making VariationPoint optional is that the aggregation *may be* variant, but does not have to be variant. Hence, if the aggregation is used in a context where there is no variability – the aggregation is always there – then there is no need to add a variation point to the model (on M1 level). This helps to reduce the complexity of the resulting model, and trims the resulting XML representation.

7.2.2 Binding Time

Within VariationPoint, the class ConditionByFormula has an attribute bindingTime which defines the *latest* binding time for this variation point. This binding time is further constrained by the tag Vh.LatestBindingTime that is attached to the aggregation:

ConditionByFormula.bindingTime $\leq$ *aggregation*.Vh.LatestBindingTime

This makes sure that the meta model can define a restriction on M2 level. The actual binding time is specified on M1 level (when the value for the bindingTime attribute is fixed).

[5] The "excess" {PartClass} elements, i.e. the "unused" variants, are deleted in the binding process.

7.2.3 Multiplicity of {PartClass}

Table 7.1 shows how the multiplicity of {PartClass} changes during the transformation.

{PartClass} annotated meta model	{PartClass} extended meta model
$0 \ldots 1$	$0 \ldots *$
$0 \ldots n$	$0 \ldots *$
$0 \ldots *$	$0 \ldots *$
$1 \ldots n$	$1 \ldots *$
$1 \ldots *$	$1 \ldots *$
$m \ldots n$	$m \ldots *$
n	$n \ldots *$
$*$	$*$

Table 7.1: Multiplicity in the Association Pattern

The change in the multiplicity means that after the transformation, the model is *less strict* than it had been before. A multiplicity that had a fixed upper bound (or was a constant) before the transformation is replaced by one that has an unlimited upper bound.

The reason for this is that we need to provide several alternative {PartClass} elements and then choose one or more of them. Hence, we need to relax the original multiplicity – otherwise there would be no way to add the additional elements. And since the number of additional {PartClass} elements to choose from cannot be known at this time, the upper multiplicity is always $*$.

7.2.4 XML Representation

The *aggregation pattern* has a low impact on the XML representation of a M1 model.

In fact, when a “normal” aggregation is made into a variation point, the only difference is that the XML element which corresponds to the {partRole} UML element gets an additional XML element named <VARIATION-POINT>.

An example for the XML code that is produced by the *aggregation pattern* can be found in Listing 7.2.

7.2.5 Notes and Restrictions

1. If an association from {WholeClass} to {PartClass} is tagged with ≪atpVariation≫, then *any* occurrence of {PartClass} may aggregate a VariationPoint. In other words, {PartClass} may aggregate VariationPoint even in cases where it is not at the end of an association that is tagged with ≪atpVariation≫.

However, using a variation in such a way would obviously not be compatible with the definition in the annotated meta model (see Figure 6.1).

2. If `{PartClass}` has a subclass (say, `SubPartClass`) that is aggregated elsewhere (say, `AnotherWholeClass`) and this other aggregation is also tagged with ≪`atpVariation`≫, then `SubPartClass` would aggregate *two* variation points.

   To avoid this, the schema generator (i.e., the metamodel tool) uses a special processing step to clean up such duplicate variation points.

3. The *aggregation pattern* cannot be used for ≪`primitive`≫ elements. This is because of the special handling of ≪`primitive`≫ elements, as defined in section 3.11 of [2].

   The obvious workaround would be to make the element non-primitive.

4. If `Vh.LatestBindingTime` is earlier than `PostBuild`, then `VariationPoint` cannot have a *PostBuild* branch, i.e. it cannot aggregate a `PostBuildVariantCondition`. This is explained in more detail in Section 7.6.6.

7.3 *Association Pattern* for Variation Points

Figure 7.2 shows how the metamodel tool transforms an association into an M2 model with variation information.

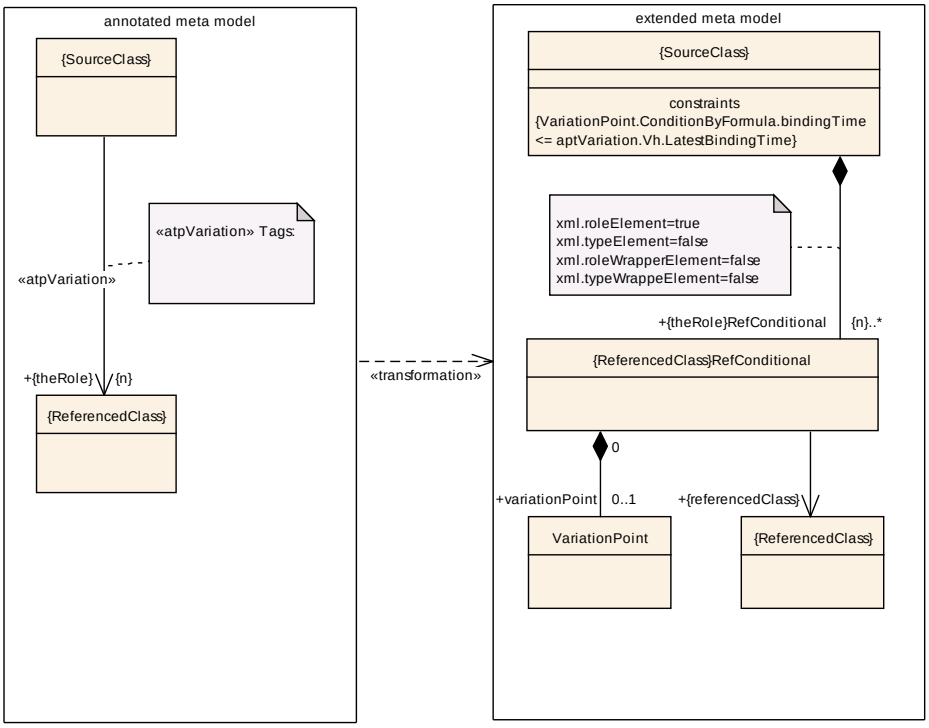

Figure 7.3: Variation Point in Association

The major difference between this pattern and the aggregation pattern (Section 7.2) is the addition of the `{ReferencedClass}RefConditional` class. The `VariationPoint` is now aggregated by `{ReferencedClass}RefConditional` instead of `{ReferencedClass}`.

7.3.1 Description

In a nutshell, the pattern for transforming an association works as follows:

1. `{SourceClass}` aggregates a `{ReferencedClass}RefConditional` element. This is very similar to the *implementation* of nonvariant associations, which introduces a `{ReferencedClass}Ref` element that acts as a pointer (read: reference) to `{ReferencedClass}`.

2. `{ReferencedClass}RefConditional` aggregates a `VariationPoint` element. The `VariationPoint` controls whether the element `{ReferencedClass}RefConditional` exists.

   The multiplicity of this aggregation is `0...1`. That is, `VariationPoint` is optional. If the `VariationPoint` element is omitted, then there is no variation, and `{ReferencedClass}RefConditional` always exists (this is equivalent to a variation point where the condition always evaluates to *true*.)

   The reason for making `VariationPoint` optional is that the association *may be* variant, but does not have to be variant. Hence, if the association is used in a context where there is no variability – the association is always there – then there is no need to add a variation point to the model (on M1 level). This helps to reduce the complexity of the resulting model, and trims the resulting XML representation.

3. `{ReferencedClass}RefConditional` provides a reference to `{ReferencedClass}`.

   As said before, the nonvariant case would use `{ReferencedClass}Ref` instead, which also has a reference to `{ReferencedClass}`, but lacks the aggregated `VariationPoint`.

7.3.2 Binding Time

The binding time for the *association pattern* follows the same schema as in the *aggregation pattern* (Section 7.2.2).

7.3.3 Multiplicity of `{ReferencedClass}RefConditional`

The multiplicity of `{ReferencedClass}RefConditional` in the *association pattern* follows the same scheme as the multiplicity of `{PartClass}` in the *aggregation pattern* (Section 7.2.3).

7.3.4 XML Representation

The *association pattern* has a low impact on the XML representation of a M1 model.

The element `{ReferencedClass}RefConditional` is visible on the XML level by its role name `{theRole}RefConditional`. This is only a slight difference from the non-variant case, which uses the role name `{theRole}Ref`. Both elements serve the same purpose: they are containers for the actual reference. The only difference is that the variant case adds an optional `VariationPoint`.

An example for the XML code that is produced by the *association pattern* can be found in Figure 7.4.

7.4 *Attribute Value Pattern* for Variation Points

Our first two patterns (Sections 7.2 and 7.3) dealt with the existence of a relationship between two elements. The pattern which will be described in this section implements a different kind of variation, namely a variation in the value of one or more attributes.

7.4.1 Description

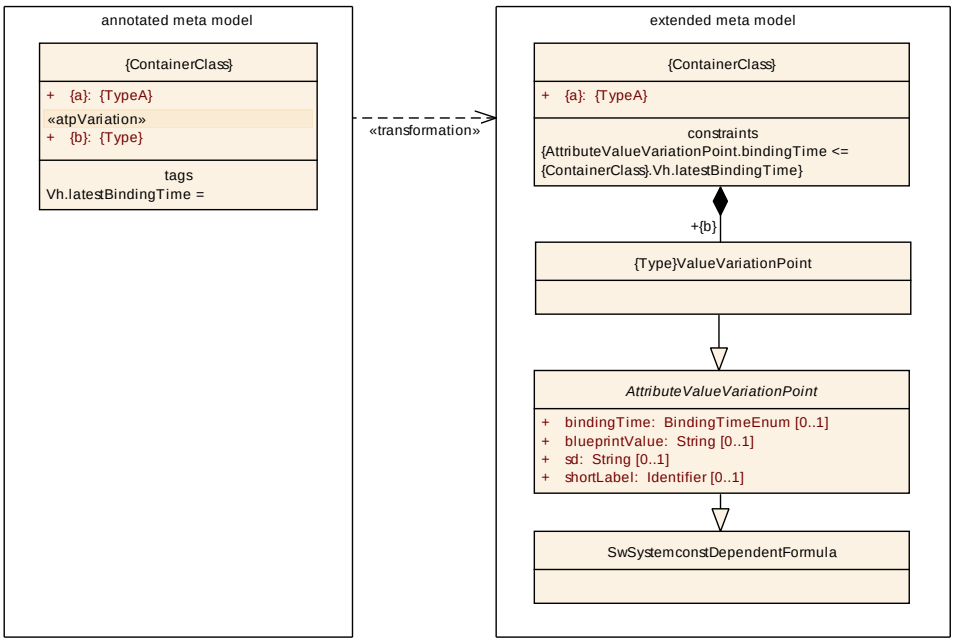

Figure 7.4: *attribute value pattern*

Class	≪`atpMixedString`≫ **AttributeValueVariationPoint (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents the ability to derive the value of the Attribute from a system constant (by SwSystemconstDependentFormula). It also provides a bindingTime.			
Base	ARObject,FormulaExpression,SwSystemconstDependentFormula			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
bindingTime	BindingTimeEnum	0..1	attr	This is the binding time in which the attribute value needs to be bound. If this attribute is missing, the attribute is not a variation point. In particular this means that It needs to be a single value according to the type specified in the pure model. It is an error if it is still a formula. **Tags:** xml.attribute=true
blueprintValue	String	0..1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:** xml.attribute=true
sd	String	0..1	attr	This special data is provided to allow synchronisation of Attribute value variation points with variant management systems. The usage is subject of agreement between the involved parties. **Tags:** xml.attribute=true
shortLabel	Identifier	0..1	ref	This allows to identify the variation point. It is also intended to allow RTE support for CompileTime Variation points. **Tags:** xml.attribute=true

Table 7.2: AttributeValueVariationPoint

In this pattern, the stereotype ≪`atpVariation`≫ marks those attributes that are variant[6]. The transformation works as follows:

- `{ContainerClass}` is stripped of all variant attributes. In Figure 7.4, this leaves only attribute `{a}`.
- For each variant attribute, an element `{Type}ValueVariationPoint` is generated. `{Type}` may be one of `Integer`, `Float`, `Boolean`, or `Numerical`, `PositiveInteger` and `NumericalInteger`.

  The class `{Type}ValueVariationPoint` is described in Section 7.4.3.

[6] In Figure 7.4, all attributes that are listed *below* ≪`atpVariation`≫ have this particular stereotype. In other words, `{a}` is a non-variant attribute, while `{b}` has the stereotype ≪`atpVariation`≫.

- `{Type}ValueVariationPoint` inherits from `AttributeValueVariationPoint`, whose attribute `bindingTime` specifies the binding time for the variant attribute (on M1 level). This is described in more detail in Section 7.4.4.
- `AttributeValueVariationPoint` in turn inherits from `SwSystemconstDependentFormula`, which implements a formula (Section 7.6.8). This formula provides the value for the variant attribute.

7.4.2 `AttributeValueVariationPoint`

`AttributeValueVariationPoint` contains four attributes, namely `bindingTime`, `shortName`, `sd`, and `blueprintValue`:

- `bindingTime` is described in Section 7.4.4.
- The `shortLabel` serves the same purpose as the `shortLabel` attribute of `VariationPoint`.

  [constr_2521] The `shortLabel` in `VariationPoint` must be unique ⌈ The `shortLabel` must be unique within the next enclosing `Identifiable`, and is used to individually address variation points in the *variant rich M1 model*. See Section 7.6.2 for details. ⌋
- The `sd` attribute is a stripped down version of the `sd` member of a `VariationPoint` (see Section 7.6.3).

  `sd` is a string that may be used by an external application to add custom data.

  There are two reasons for not using a special data group like in `VariationPoint`. First, a variation point for an attribute value is not as structurally significant as one for full element, so it is conceivable that there less data are needed.

  Second, if `AttributeValueVariationPoint` would aggregate a special data group, then the resulting XML representation would require an additional wrapper – even though the aggregation is optional. This would be a significantly higher overhead than in the *aggregation pattern* and in the *association pattern*.
- The `blueprintValue` is used if the variation point is part of a blueprint. It contains a description which provides instructions how to derive appropriate objects from the blueprint. For more details on variation points in blueprints, see Section 7.6.11 and [9].

  [constr_2567] Undefined Value in Attribute Value Blueprints ⌈ If a `blueprintValue` is specified, then the `value` defined by the `AttributeValueVariationPoint` is not used and should therefore at least contain one term `undefined` which is to be refined when deriving objects from this blueprint. ⌋

Both `shortLabel` and `sd` are optional. `bindingTime` may be omitted under certain circumstances as described in Section 7.4.4. `blueprintValue` is only allowed in blueprints and may not be present in a system description.

7.4.3 {Type}ValueVariationPoint

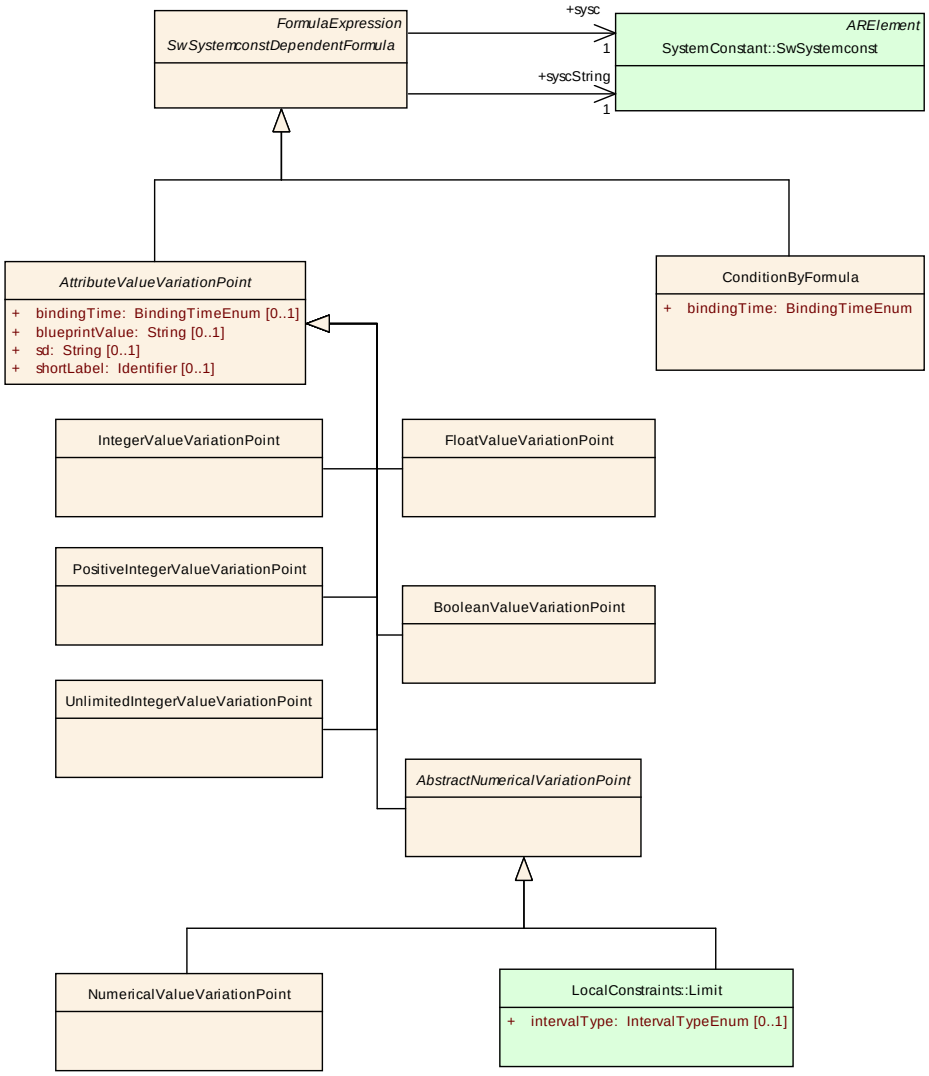

Figure 7.5: {Type}ValueVariationPoint

As Figure 7.5 illustrates, the class `AttributeValueVariationPoint` comes in six different flavors, namely `IntegerValueVariationPoint`, `FloatValueVariationPoint`, `BooleanValueVariationPoint`, `NumericalValueVariationPoint`, `PositiveIntegerValueVariationPoint` and `UnlimitedIntegerValueVariationPoint`.

The reason for adding these extra classes is as follows. We could have defined the *attribute value pattern* without them, by just letting `{ContainerClass}` (Figure 7.4) aggregate a `AttributeValueVariationPoint`. But then, any trace of the *type* of the original attribute would have been lost.

7.4.4 Binding Time

The binding time for attribute values is specified in the attribute `bindingTime` of `AttributeValueVariationPoint`. Each attribute may have its own binding time.

The attribute `bindingTime` may be omitted if `SwSystemconstDependentFormula` specifies a single constant value for the attribute, and not an expression[7]. In this case, the value is fixed from the start. In all other cases, the attribute `bindingTime` must be present.

The attribute `bindingTime` is further constrained by the tag `Vh.LatestBindingTime` that is attached to the `{ContainerClass}` on the left side in Figure 7.4

$$\texttt{bindingTime} \leq \texttt{\{ContainerClass\}.Vh.LatestBindingTime}$$

Furthermore, the tag `Vh.LatestBindingTime` is limited to `PreCompileTime` and earlier[8] in this pattern.

7.4.5 Multiplicity of `AttributeValueVariationPoint`

In an M1 AUTOSAR model, one or more `{Type}ValueVariationPoint` elements must be created for each variant attribute in the original `{ContainerClass}`. The exact number is determined as follows:

- If the original attribute had a multiplicity of 1, then exactly one `{Type}ValueVariationPoint` is generated.
- If the original attribute was an array (i.e., it had a multiplicity of $[0 \ldots *]$, $[1 \ldots *]$ or more generally $[m \ldots n]$, then as many `{Type}ValueVariationPoint`s must be created as there are entries in the array.

  In this case, the *sequential order* of the corresponding XML elements is significant and must correspond to the actual succession of elements in the array.

7.4.6 XML Representation

Without variability, both the attributes `{a}` and `{b}` in Figure 7.4 would be represented in the XML schema as individual elements, each holding a constant value.

The *attribute value pattern* replaces variant attributes with a system constant expression. If this expression is just a value, then the resulting XML has the same structure as in the non-variant case (except for the binding time, but even this attribute may be

[7] That is, if the formula does not contain any references to other system constants, nor any functions or operators, just a single value.

[8] That is, `SystemDesignTime`, `CodeGenerationTime` and `PreCompileTime`.

omitted for such expressions). A complicated expression may of course add significant overhead compared to the non-variant case.

The `shortLabel` and `sd` (and `bindingTime` for a single-value expression) are optional and do not add any overhead to the XML representation if they are not present.

An example for the XML code that is produced by the *attribute value pattern* can be found in Figure 7.6.

7.4.7 Notes and Restrictions

1. The binding time for an `AttributeValueVariationPoint` is at most *PreCompileTime*. We do not support to use such a variation point with a `PostBuild` binding time. This is because such a behavior is already covered by calibration parameters.
2. It is not possible to model the *existence* of an attribute.

   The obvious way to do this would be to move the attribute in question to a separate element, aggregate this element and then make the aggregation a variation point (see *aggregation pattern*, Section 7.2).
3. Since each element of an array is specified with a separate `AttributeValueVariationPoint` element, it is no longer mandatory that all array elements have the same binding times. However, it is considered good practice to use identical binding times for all array elements.
4. The *attribute value pattern* is only defined for attributes of type `Integer`, `Float`, `Boolean`, `Numerical`, `PositiveInteger` and `NumericalInteger`. Any extension to other types – as long as they are covered by the expression language – would require a change in the meta model.
5. The rationale for making `bindingTime` optional is that this makes the XML representation simpler. This is especially helpful for `EvaluatedVariantSet`s, which use the *attribute value pattern* to define values for system constants. In a typical use case, these values are constants, not formulas.

7.5 *Property Set Pattern* for Variation Points

Like the previous pattern, this one also deals with variations in attributes. However, the pattern introduced in Section 7.4 requires that every variant attribute is annotated with a stereotype ≪`atpVariation`≫. This also means that the M2 meta model has to decide which attribute is variant, and which is not. Such a decision is not always practical, e.g. when there are a large number of attributes, each of which may be subject to variation.

The pattern which we define in this section follows a different approach. By applying the stereotype ≪atpVariation≫ to the *class* that contains the attributes (and not to individual attributes of the class), we can define all attributes as potentially variable. Attributes are then partitioned into several sets of variant attributes, each of which has a variation point.

7.5.1 Example

Consider the following example. A class named PropertiesClass has six attributes: attr1, attr2, attr3, attr4, attr5, and attr6. Of these, the first four attributes are variant, while attr5 and attr6 are invariant.

Now, one solution would be to apply ≪atpVariation≫ to the attr1, attr2, attr3, and attr4. In this case, each attribute would be its own variation point, and could be varied independently.

However, it might be that there are three different sets of values for attr1, attr2, attr3, and attr4:

	attr1	attr2	attr3	attr4
Set1	1	1	2	2
Set2	2	3	4	1
Set3	3	1	6	3

Table 7.3: Example for *Property Set Pattern*

The *property set pattern* allows us to specify these three sets. When the stereotype ≪atpVariation≫ is applied to PropertiesClass (in the annotated meta model), the class is transformed (in the extended meta model) into a new PropertiesClass which has *no* attributes, but aggregates one or more classes named PropertiesClassConditional which now contain the attributes[9]. Each PropertiesClassConditional aggregates a variation point, very much like {PartClass} in the aggregation pattern (see Section 7.3).

The idea is that a concrete model will contain one instance of PropertiesClass (without attributes), which aggregates *four* (not three!) instances of PropertiesClassConditional. The first three instances hold Set1, Set2, and Set3, respectively. The remaining instance of PropertiesClassConditional contains the invariant attributes, namely attr5 and attr6. Their VariationPoints must have appropriate conditions that make sure that exactly one of the first three instances PropertiesClassConditional can be selected. The last PropertiesClassConditional may omit its VariationPoint because the attributes that are defined there are invariant.

How can it be that the first three PropertiesClassConditional hold different attributes than the last one? Actually, PropertiesClassConditional contains a

[9] As we will see in the next section, the actual transformation is a bit more involved, but the general idea is the same as outlined in this example.

copy of all attributes that were in the original `PropertiesClass` (in the annotated meta model), but the lower multiplicity of each attribute is reduced to `0` – that is, all attributes are optional. This way, each instance of `PropertiesClassConditional` may hold an arbitrary subset of the original attributes. The condition is that after the variation has been bound, the *sum* of all attributes must be equal to the original set of attributes[10].

The property set pattern is actually a bit more flexible than the example shows. First, the variant attributes do not need to be in the same set; they can also be distributed over several `PropertiesClassConditional` classes. Second, the pattern not only applies to attributes, but also to aggregated and associated elements. Third, the pattern may be combined with the attribute value pattern (and others) to create even more powerful variant structures.

7.5.2 Description

The transformation that is involved with the *property set pattern* is more complex than the one for the previous patterns. This time, we apply the stereotype ≪`atpVariation`≫ to a class, not a relation or an attribute. Unlike the other patterns, a class may inherit its stereotype from another class (or may aggregate other classes), which requires special consideration.

Hence, we describe the *property set pattern* in two steps. First, we show how the transformation works for a class which is explicitly tagged in the meta model with a stereotype ≪`atpVariation`≫ and does not derive from a class that has the same stereotype. In a second step, we extend this description to show what happens if a class is not explicitly tagged with ≪`atpVariation`≫, but is *derived* from such a class and therefore implicitly has the stereotype.

[10] For completeness: attributes that were optional in the pure meta model may still be omitted.

7.5.2.1 ≪atpVariation≫ Applied Directly to a Property Set Class

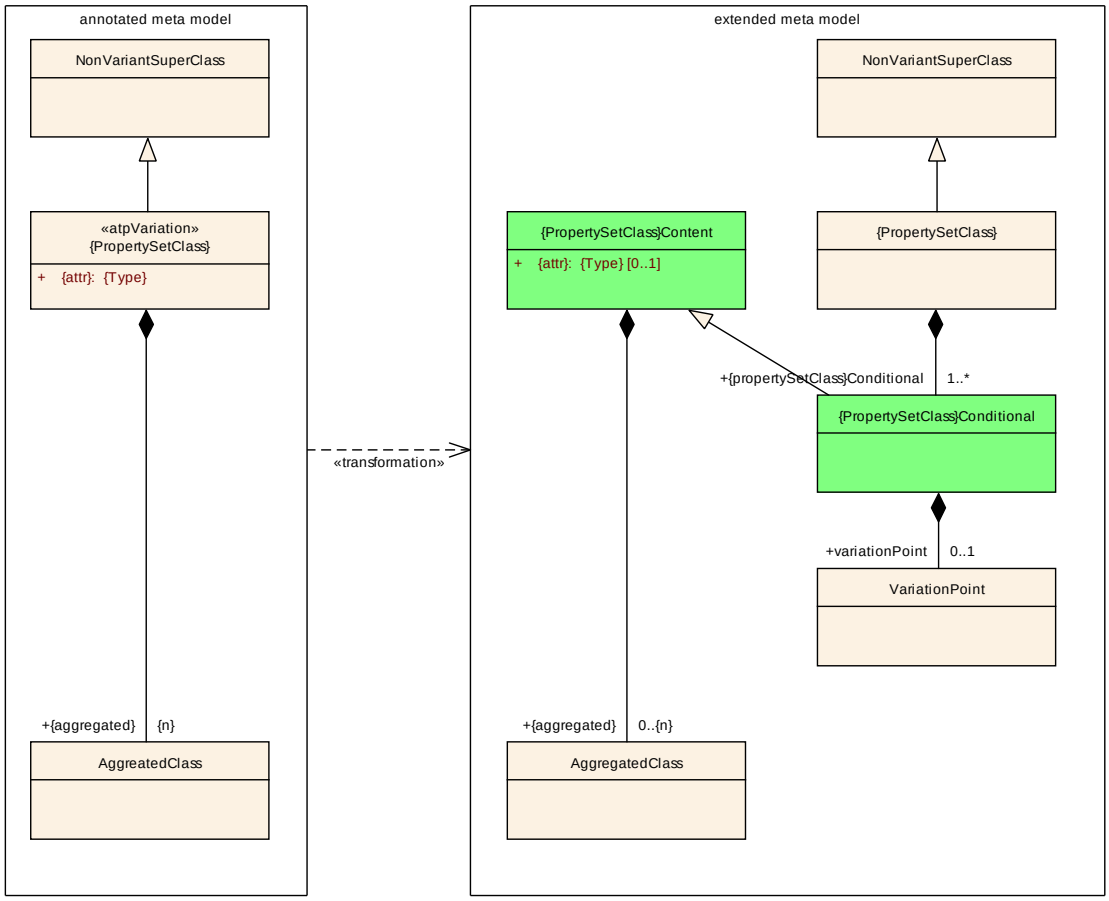

Figure 7.6: *Property Set Pattern* without inheritance in {PropertySetClass}

The annotated meta model for this transformation is centered around {PropertySetClass}:

1. {PropertySetClass} is based on NonVariantSuperClass, which will not be changed by this pattern.
2. Class {PropertySetClass} has the stereotype ≪atpVariation≫ and contains an attribute named {attr}.
3. {PropertySetClass} aggregates AggregatedClass. AggregatedClass itself is not changed when the pattern is applied, although it will be aggregated by a different class afterwards.

The transformation works as follows:

1. All attributes and the stereotype ≪atpVariation≫, are removed from {PropertySetClass}.
2. The transformation generates a new class {PropertySetClass}Content which contains the original attribute(s) of {PropertySetClass}, but now with a lower multiplicity of 0.

The change in the multiplicity stems from the fact that this class is designed to hold an arbitrary subset of the attributes that were originally in {PropertySetClass}.

{PropertySetClass}Content can be seen as a kind of a clone of {PropertySetClass} that lacks the inheritance of NonVariantSuperClass and has a different multiplicity for its attribute.

3. A new class named {PropertySetClass}Conditional is generated. {PropertySetClass}Conditional derives from {PropertySetClass}Content, and aggregates a VariationPoint.

   Because it inherits from {PropertySetClass}Content, each instance of {PropertySetClass}Conditional holds a subset of the attributes that were originally in {PropertySetClass} in the *annotated meta model*.

4. {PropertySetClass} aggregates an arbitrary number of {PropertySetClass}Conditional elements. The idea is that a particular instance of {PropertySetClass}Conditional in the *extended meta model* contains a subset of the original attributes of {PropertySetClass} in the *annotated meta model*. But *after the binding*, the disjoint sum of all attributes given in {PropertySetClass}Conditional instances must yield the full set of attributes that were in {PropertySetClass} in the *annotated meta model*.

5. The VariationPoint finally decides whether a {PropertySetClass}Conditional is included in a particular variant. That is, the aggregation from {PropertySetClass} to {PropertySetClass}Conditional is itself subject to variation.

6. AggregatedClass is now aggregated by {PropertySetClass}Content, with a lower multiplicity of 0. The reason for the change in the multiplicity is that aggregated classes are handled in the same way as attributes.

7. (Not shown in the above diagram) References to {PropertySetClass} in the annotated meta model still point to {PropertySetClass} in the extended meta model.

7.5.2.2 ≪atpVariation≫ Applied to Superclass of a Property Set Class

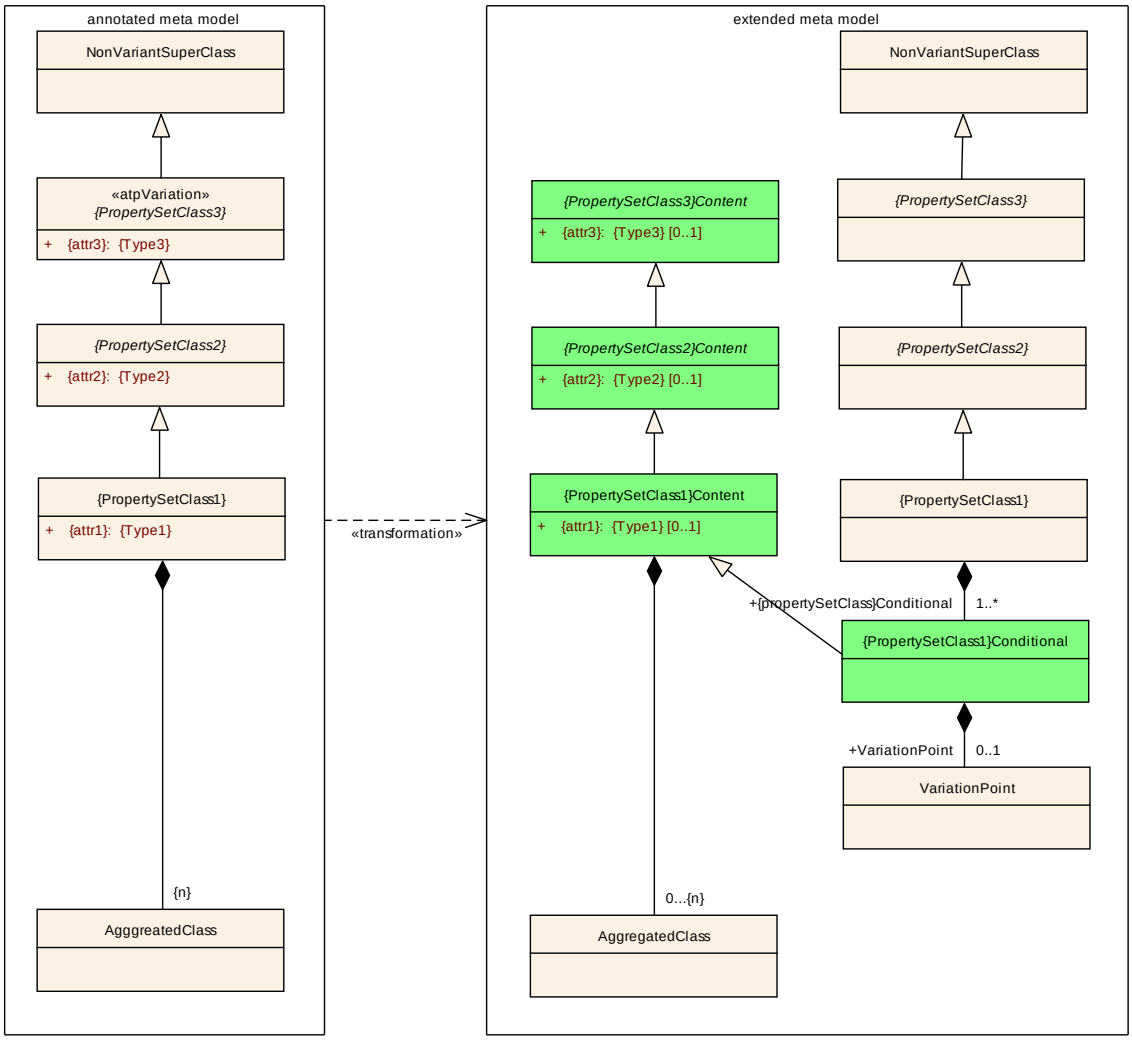

Figure 7.7: *Property Set Pattern* with inheritance in `{PropertySetClass}`

When the stereotype ≪atpVariation≫ is applied to a superclass of the property set class, then the situation becomes slightly more complex:

1. `NonVariantSuperClass` and `AggregatedClass` are the same as before.
2. `{PropertySetClass1}` plays the role that was occupied by `{PropertySetClass}` in the previous section. However, `{PropertySetClass1}` now derives from a class `{PropertySetClass3}`, which is also tagged with the stereotype ≪atpVariation≫. `{PropertySetClass2}` sits between those two classes.

The actual transformation does not differ much between Figures 7.7 and 7.6. The only new aspect in Figure 7.7 is that for each property set class *up to* `{PropertySetClass3}`, we create a hierarchy of `{PropertySetClass}Content` classes. These classes retain the attributes of the original classes, as well as their aggregations and association relations.

7.5.2.3 Constraints

The property set pattern needs a constraint that makes sure that for any variant, the set of all attribute values defined with this method is complete *and* there are no double definitions.

[constr_2506] Attributes in property set pattern ⌈ On M1 level, let C be the set of attributes (or aggregated elements[11]) that would have been in the original[12] `{PropertySetClass}` object, and $C_1, \ldots, C_n$ be the respective sets of attributes in the `{PropertySetClass}Conditional` objects **for a given variant**. Also, let C' be the set of non-optional attributes, e.g., those with a lower multiplicity of 1.

We define the following constraints:

$$\forall C_i, C_j \text{ in the given variant} : C_i \cap C_j = \emptyset$$
$$C' \subseteq C_1 \cup C_2 \cup \ldots C_n \subseteq C$$

⌋

One might wonder why there is no class named `PropertySetClass_invariant` for those attributes that never change. The reason is that such a class can easily be realized by using a `{PropertySetClass}Conditional` which does *not* aggregate `VariationPoint`. Such a `{PropertySetClass}Conditional` is nonvariant, i.e. it always exists.

7.5.3 Binding Time

The latest binding time for the *property set pattern* is *PostBuild*.

7.5.4 Multiplicity of Attributes and aggregated elements

In the *property set pattern*, attributes (and aggregated elements) are moved from `{PropertySetClass}` in the *annotated meta model* to `{PropertySetClass}Conditional` in the `extended meta model`.

With this move, the lower multiplicity always changes to 0.

7.5.5 XML Representation

An example for the XML code that is produced by the *property set pattern* can be found in Figure 7.7.

[11] The constraints defined in this section apply to attributes as well as aggregates elements, due to the close relationship of the two in the AUTOSAR meta model. For simplicity, the rest of this section talks about “attributes” only.

[12] In this context, “original” means `{PropertySetClass}` without the stereotype ≪`atpVariation`≫. In other words, “original” means “as in the pure meta model”.

Despite the perceived complexity of the pattern in Figures 7.6 and 7.7, the impact of the *property set pattern* on the XML code is rather limited. As Example 7.7 shows, the property set pattern adds a `-VARIANTS` wrapper around the attributes, and a `-CONDITIONAL` element for each (sub)set of attribute values. `-CONDITIONAL` also contains a `VARIATION-POINT` element.

So, the main impact on the XML code is the duplication of attribute values, but the overhead introduced by variant handling should only add a few elements.

7.5.6 Comparison with Other Patterns

Both this and the *attribute value pattern* (Section 7.4) are aimed at attributes, but with several differences:

- The *prototype set pattern* provides a way to *group* attributes that belong together.
- The *property set pattern* is more flexible in that variability is not restricted to those attributes for which the M2 meta model "allows" variability. There is however a catch: because of the higher flexibility, it is not a priori clear which attributes will be invariant, and which not.
- The *attribute value pattern* may use an expression to define the value of an attribute, while the *property set pattern* can only use a fixed value (more precisely, a fixed value *per variant*). However, the *property set pattern* may be combined with the *attribute value pattern* to achieve this effect.
- The *attribute value pattern* is available for a limited number of data types only, namely `Integer`, `Float`, `Boolean`, `Numerical`, `PositiveInteger` and `NumericalInteger`. The *property set pattern* has no such limitation.
- If an attribute is optional, then the *property set pattern* may also decide whether an attribute exists or not. This is different from the *attribute value pattern*, which can only change the value of an attribute.

Furthermore, the *Property Set Pattern* differs from the *Aggregation Pattern* and the *Association Pattern* in that the former pattern works on a number of attributes, aggregations and associations at once, while the latter patterns determine the existence of a singe aggregation or aggregation only.

7.5.7 Combining the *attribute value pattern* and the *property set pattern*s

In the previous section, we said that the *property set pattern* cannot specify an expression to define the value of an attribute. While this is true, there is a way to avoid this restriction, namely by combining the *property set pattern* with the *attribute value pattern*.

In this case, the *property set pattern* would allow to partition the complete set of attributes into several disjoint subsets. Each of these individual attributes may have the stereotype ≪atpVariation≫, which means that the *attribute value pattern* is applied, and the value of the attribute may be determined by an expression.

Furthermore, since an attribute may occur in multiple sets (on M1 level), there may be multiple expressions for determining the value of a particular attribute, each tailored for a particular variant.

7.6 VariationPoint

The structure of a VariationPoint is illustrated in Figure 7.8.

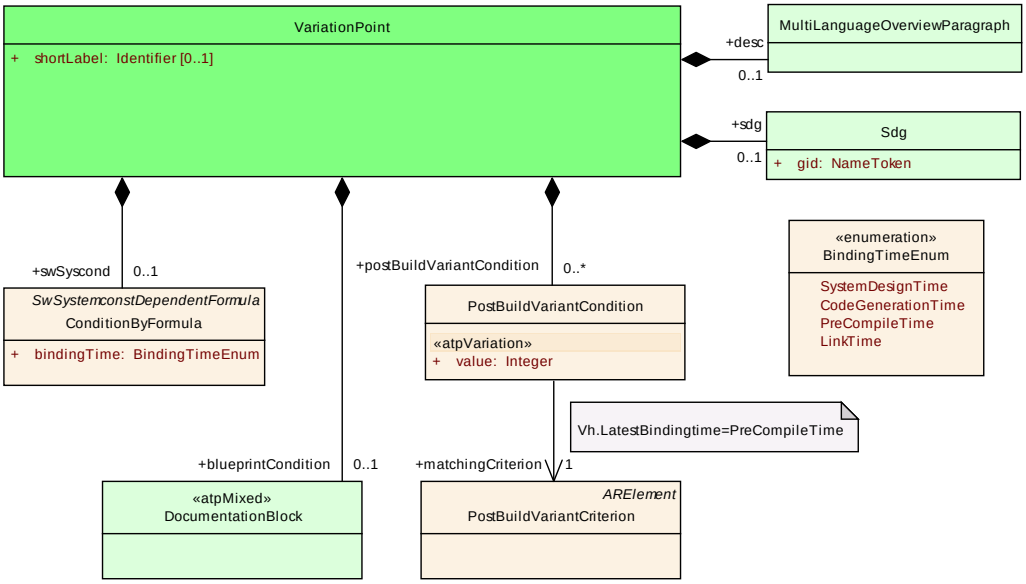

Figure 7.8: Variation Point

Class	**VariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to express a "structural variation point". The container of the variation point is part of the selected variant if swSyscond evaluates to true and each postBuildVariantCriterion is fulfilled.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
desc	MultiLanguage OverviewParagraph	0..1	aggr	This allows to describe shortly the purpose of the variation point. **Tags:** xml.sequenceOffset=20

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
blueprintCondition	DocumentationBlock	0..1	aggr	This represents a description that documents how the variation point shall be resolved when deriving objects from the blueprint. Note that variationPoints are not allowed within a blueprintCondition. **Tags:** xml.sequenceOffset=28
postBuildVariantCondition	PostBuildVariantCondition	*	aggr	This is the set of post build variant conditions which all shall be fulfilled in order to (postbuild) bind the variation point. **Tags:** xml.sequenceOffset=40
sdg	Sdg	0..1	aggr	An optional special data group is attached to every variation point. These data can be used by external software systems to attach application specific data. For example, a variant management system might add an identifier, an URL or a specific classifier. **Tags:** xml.sequenceOffset=50
shortLabel	Identifier	0..1	ref	This provides a name to the particular variation point to support the RTE generator. It is necessary for supporting splitable aggregations and if binding time is later than CodeGenerationTime, as well as some RTE conditions. It needs to be unique with in the enclosing Identifiables with the same ShortName. **Tags:** xml.sequenceOffset=10
swSyscond	ConditionByFormula	0..1	aggr	This condition acts as Binding Function for the VariationPoint. Note that the mulitplicity is 0..1 in order to support pure postBuild variants. **Tags:** xml.sequenceOffset=30

Table 7.4: VariationPoint

7.6.1 The structure of class `VariationPoint`

The class `VariationPoint` holds information about a variation point in the *aggregation pattern* (Section 7.2), the *association pattern* (Section 7.3), and the *property set pattern* (Section 7.5)[13].

A `VariationPoint` aggregates a `ConditionByFormula`, a `PostBuildVariantCondition` and a `DocumentationBlock` in the role `blueprintCondition`. These

[13] The *attribute value pattern* (Section 7.4) is simpler and does not make use of the class `VariationPoint`. Furthermore, its latest binding time is `CompileTime`, so many of the issues discussed in this section do not apply to this pattern.

three “branches” are independent of each other. As the multiplicities in Figure 7.8 shows, they are also all optional:

- If a variation point aggregates `ConditionByFormula`, then this variation point is a *PreBuild* variation point. See Section 7.6.5 for details.
- If a variation point aggregates a `PostBuildVariantCondition`, then this variation point is a *PostBuild* variation point. See Section 7.6.6 for details.
- If a variation point aggregates a `DocumentationBlock` in the role `blueprintCondition`, then this variation shall be resolved when deriving objects. Refer to [9] for details.
- A variation point may also aggregate *both* `ConditionByFormula` and `PostBuildVariantCondition`. In this case, it is both a *PreBuild* and a *PostBuild* variation point. See Section 7.6.5 for details.
- Technically, a variation point may also aggregate *none* of the above classes. In this case, there is no variation at all, and the element to which the variation point is attached to always exists.

  This is equivalent to a *PreBuild*-only variation point where `ConditionByFormula` has binding time `SystemDesignTime`, and who’s formula always evaluates to *true*.

In all patterns, the `VariationPoint` element has a multiplicity of $[0 \ldots 1]$, that is, it is optional. If the variation point is omitted, then there is no variation and the respective element always exists.

[constr_2557] no `VariationPoints` where `Vh.latestBindingTime` set to `BlueprintDerivationTime` in system configurations ⌈ Blueprints are **not** part of a system configuration. In consequence of this, in a system configuration there shall be no `VariationPoint` where `Vh.latestBindingTime` is restricted to `BlueprintDerivationTime` by the meta model. ⌋

[constr_2558] If `Vh.latestBindingTime` is `BlueprintDerivationTime` then there shall only be `blueprintCondition/blueprintValue` ⌈ `VariationPoints` with `latestBindingTime` restricted to `BlueprintDerivation` shall not have `swSysCond` nor `postbuildVariantCondition`. ⌋

[constr_2559] no nested `VariationPoint` ⌈ As `blueprintCondition` is a `DocumentationBlock` it could again contain `VariationPoints` and therefore would allow nesting of `VariationPoints`. This is not intended and shall not be used. ⌋

7.6.2 `shortLabel`

`VariationPoint` has a single optional attribute `shortLabel` that implements a name for the variation point.

[constr_2514] `shortLabel` in `VariationPoint` must be unique ⌈ `shortLabel` must be unique with the next enclosing `Identifiable` with the same `ShortName`. ⌋

For example, in the aggregation pattern (Section 7.2), this enclosing `Identifiable` as usually `{WholeClass}`.

Normally, AUTOSAR would use the attribute `ShortName` of the next enclosing `Identifiable` as a unique name. This does not work with variation points. The reason for this is rooted in Figure 6.1, more precisely in the difference between the *variant rich M1 model* and the *bound M1 model*.

The *variant rich M1 model* (see Figure 6.1) may define several alternative variants for one aggregation. As the term "alternatives" implies, only one of them is left in the *bound M1 meta model*.

But the `ShortName` does not provide sufficient capabilities for unique identification in this case: since we don't know which alternative "survives", all have to use the same `ShortName`. Otherwise, it would not be possible to identify the "surviving" alternative in the *bound M1 model* by `ShortName` because we do not know the correct `ShortName` before binding.

Strictly speaking, a (not-yet-bound) *variant rich M1 model violates* AUTOSAR's consistency conditions by having multiple elements with the same `ShortName`. This is only feasible because we require that the *bound M1 model* (and the associated code) will eventually adhere to those consistency rules ([constr_2503]).

There are several situations where it is necessary to individually address variations in the *variant rich M1 model* that have the same `ShortName`:

- If an aggregation has the stereotype ≪`atpSplitable`≫. The use case for this is that particular variants are held in a separate artifact. In order to merge such separate artefacts, it is necessary not only to consider `shortName` but also the particular variants of an Identifiable (in particular the `shortLabel`. For more details about splitable elements refer to Section 2.3.2.
- If binding time is `CodeGenerationTime` or later, the RTE needs to distinguish between the individual variants if `PreCompileTime` variability is implemented.
- It is often necessary to refer to individual variation point from the *outside*. For example, a configuration management system might need to identify individual variation points for traceability.

Also, since `shortLabel` is an optional element, it has no impact on the size or complexity of the XML if it is not present.

[constr_2512] `shortName` uniqueness constraint for variants ⌈ `shortName` + `shortLabel` of a variant element must be unique within the name space established by the surrounding `Identifiable`. ⌋

[constr_2513] splitted variants must have a `shortLabel` ⌈ If "splittable" elements with `VariationPoint`s are held in different artefacts they must have a `shortLabel` in the `VariationPoint`. ⌋

This constraint makes sure that proper merging of the model from the artifacts is supported. Note that the splitkey usually contains the `shortLabel` of the `VariationPoint`.

7.6.3 sdg

The class `VariationPoint` aggregates an optional `sdg` object (see Section 4.6) which can be used by external software systems to attach application specific data to a variation point. For example, a variant management system might add an identifier, an URL or a specific classifier to a variation point.

Since such data is highly application and vendor specific, it cannot be standardized, and a special data group is necessary instead.

Also, since `sdg` is an optional element, it has not impact on the size or complexity of the XML representation if it is not present.

7.6.4 (Latest) Binding Time

In Section 7.1, we have seen that each variation point has a binding time. Binding times (Section 7.10) can be further categorized as *PreBuild* and *PostBuild* binding times:

PreBuild This category contains the following binding times: `SystemDesignTime`, `CodeGenerationTime`, `PreCompileTime`, and `LinkTime`.

A concrete variation point (i.e., on M1 level) is subject to *PreBuild* variation if it contains a `ConditionByFormula` element (see Section 7.6.5. Its binding time is specified in the `bindingTime` attribute of the `ConditionByFormula` element (see Section 7.6.5.

PostBuild This category contains only a single binding time, namely `PostBuild`.

A concrete variation point (i.e., on M1 level) is *PostBuild* if it contains a `PostBuildVariantCondition` element (see Section 7.6.6). Since there is only one binding time for *PostBuild*, no particular attribute for specifying the binding time is necessary.

The binding time is further constrained by the tag `Vh.LatestBindingTime` that was introduced in the patterns earlier in this chapter:

- If `Vh.LatestBindingTime` = `PostBuild`, then a variation point on M1 level may have any binding time. It may be a *PreBuild* or a *PostBuild* variation point (or both, see Section 7.6.9), and may aggregate `ConditionByFormula` or `PostBuildVariantCondition`.
- If `Vh.LatestBindingTime` < `PostBuild`, then a variation point on M1 level can only be a *PreBuild*, but *not* a *PostBuild* variation point. Obviously, it may only aggregate a `ConditionByFormula` in this case.

It is obvious that the binding time of a *PreBuild* variation point (that is, the value of the attribute `bindingTime` of `ConditionByFormula`), must never exceed `Vh.LatestBindingTime`.

7.6.5 *PreBuild* Variation Points

Class	≪**atpMixedString**≫ **ConditionByFormula**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents a condition which is computed based on system constants according to the specified expression. The expected result is a considered as boolean value. The result of the expression is interpreted as a condition. • "0" represents "false"; • a value greater than zero is considered "true"			
Base	ARObject,FormulaExpression,SwSystemconstDependentFormula			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
bindingTime	BindingTimeEnum	1	attr	This attribute specifies the point in time when condition may be evaluated at earliest. At this point in time all referenced system constants must have a value. **Tags:** xml.attribute=true

Table 7.5: ConditionByFormula

All the information that is necessary to implement a *PreBuild* variation point is provided by the class `ConditionByFormula`:

- `ConditionByFormula` derives from `SwSystemconstDependentFormula`. This class implements the (boolean) formula that determines whether the variation point is “on” or “off”.

  The formula language is defined in Section 4.8. See also Section 7.6.8 for further explanation how formulas are used in the variant handling concept.

- `ConditionByFormula` has a single attribute, `bindingTime`, which defines the latest binding time for this variation point. The binding times are described in more detail in Section 7.10.

A concrete software system *may* bind a variation point at an earlier binding time if this is technically feasible, and within contractual limits[14]. We define the additional restriction

> Variation Point Binding Times later than System Design Time are part of the contract.

[14] For a definition of contract phases, see Chapter 3.1 in the RTE specification [14].

According to this restriction, the RTE Generator is not allowed to resolve the variability in the application header file even if the variability is already chosen in the input of the RTE. If the binding time is *SystemDesignTime*, then the variability is not considered part of the contract phase, and it must be bound properly during *SystemDesignTime* before the contract.

7.6.6 *PostBuild* Variation Points

Class	**PostBuildVariantCondition**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies the value which must be assigned to a particular variant criterion in order to bind the variation point. If multiple criterion/value pairs are specified, they all must must match to bind the variation point. In other words binding can be represented by `(criterion1 == value1) && (condtion2 == value2) ...`			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
matchingCriterion	PostBuildVariantCriterion	1	ref	This is the criterion which needs to match the value in order to make the PostbuildVariantCondition to be true.
value	Integer	1	attr	This is the particular value of the post-build variant criterion. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 7.6: PostBuildVariantCondition

Class	**PostBuildVariantCriterion**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies one particular PostBuildVariantSelector. **Tags:** atp.recommendedPackage=PostBuildVariantCriterions			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
compuMethod	CompuMethod	1	ref	The compuMethod specifies the possible values for the variant criterion serving as an enumerator.

Table 7.7: PostBuildVariantCriterion

A *PostBuild* variation point consists of a `PostBuildVariantCondition`, which has a reference to a `PostBuildVariantCriterion`. Unlike a *PreBuild* variation point, which is governed by a formula defined in `ConditionByFormula`, a *PostBuild* varia-

tion point is “enabled” if the value defined in PostBuildVariantCondition matches the one in PostBuildVariantCriterion.

[constr_2517] postbuildVariantCondition only for PostBuild ⌈ Aggregation of PostBuildVariantCondition in VariationPoint is only allowed if the annotated model states Vh.latestBindingTime to PostBuild. ⌋

Note that the attribute value of PostBuildVariantCondition is subject to *PreBuild* variation. It uses the *attribute value pattern*, hence its latest binding time is PreCompileTime. That is, the value which will be compared with the contents of PostBuildVariantCriterion is computed[15] at PreCompileTime (at most).

The actual comparison with the contents of PostBuildVariantCriterion, however, is done during startup.

A VariationPoint element may aggregate any number of PostBuildVariantCondition elements. A logical *and* is implied between all these elements: the variation point is “enabled” if and only if *all* the values in the aggregated PostBuildVariantCondition elements match their PostBuildVariantCriterion.

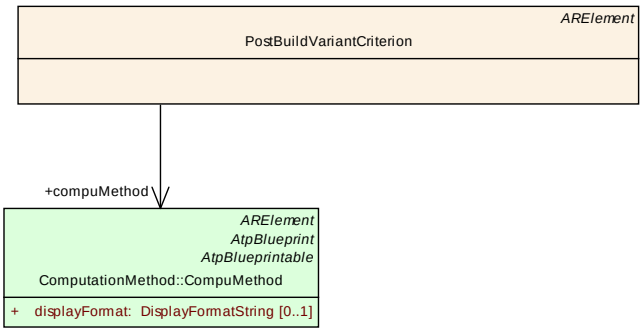

Figure 7.9: PostbuildVariantCriterion

A PostBuildVariantCriterion also contains a CompuMethod which which specifies the conversion between the physical and the internal representation of data (see the *Software Component Template* [15] for a detailed description of CompuMethod).

The RTE is responsible for managing the PostBuildVariantCriterion values.

7.6.7 System Constants

For *PreBuild* variation points, the binding function depends on SwSystemconst. Such a system constant is basically a name/value pair. shortName, dataConstr and compuMethod for a system constant are defined in SwSystemconst.

Note that compuMethod in the SwDataDefProps of SwSystemconst is intended only to support appropriate representation of the values in tools and documentation. The values shall always be set as internal value.

[15] It may be an expression rather than a constant value or a single system constant.

For more details please refer to [15].

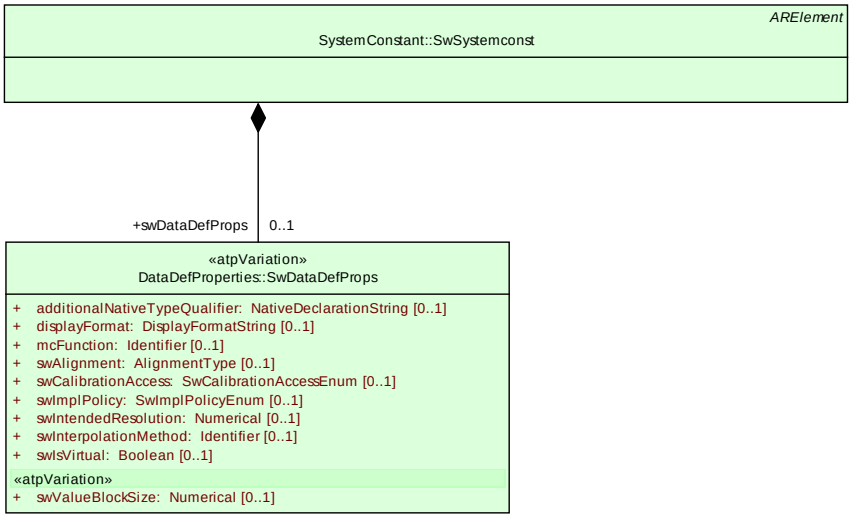

Figure 7.10: Defintion of a `SwSystemconst`

Class	**SwSystemconst**			
Package	M2::AUTOSARTemplates::CommonStructure::SystemConstant			
Note	This element defines a system constant which serves an input to select a particular variation point. In particular a system constant serves as an operand of the binding function (swSyscond) in a Variation point. Note that the binding process can only happen if a value was assigned to to the referenced system constants. **Tags:** atp.recommendedPackage=SwSystemconsts			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
swDataDefProps	SwDataDefProps	0..1	aggr	This denotes the data defintion properties of the system constant. In particular it is the limits and - in case the system constant is an enumeration - the compu method. **Tags:** xml.sequenceOffset=40

Table 7.8: SwSystemconst

In order to choose variants, values need to be assigned to `SwSystemconst`. Note that the values shall always be specified as "internal values". This is done in `SwSystemconstValue`. For more details refer to Section 7.8.

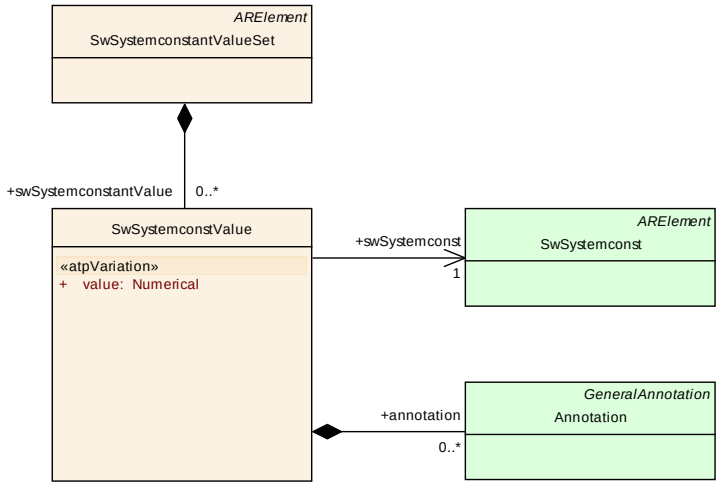

Figure 7.11: Assigning a value to a `SwSystemconst`

Class	**SwSystemconstValue**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class assigns a particular value to a system constant.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
annotation	Annotation	*	aggr	This provides the ability to add information why the value is set like it is. **Tags:** xml.sequenceOffset=30
swSystem const	SwSystemconst	1	ref	This is the system constant to which the value applies. **Tags:** xml.sequenceOffset=10
value	Numerical	1	attr	This is the particular value of a system constant. It is specified as Numerical. Further restrictions may apply by the definition of the system constant. The value attribute defines the internal value of the SwSystemconst as it is processed in the Formula Language. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=20

Table 7.9: SwSystemconstValue

7.6.8 Application of Formulas in Variation Points

Binding of variation points is performed by evaluating the formula in the variation point. These formula can be one of the subclasses according to Figure 7.12.

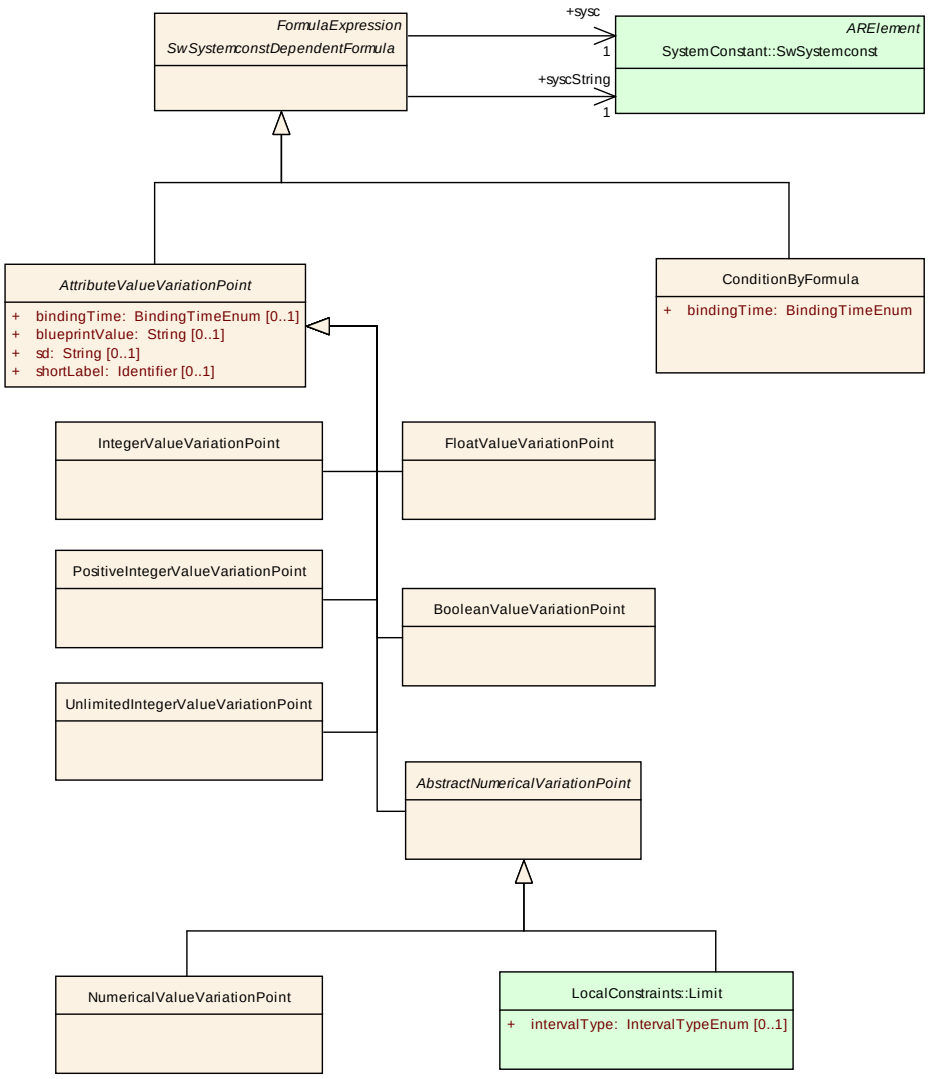

Figure 7.12: SwSystemconstDependentFormula

A `SwSystemconstDependentFormula` element is a formula which uses system constants (as indicated in Figure 7.12 by the reference `sysc` to `SwSystemconst`) as operands. Note that the multiplicity of `1` in the diagram is a technicality; a formula may actually use more than one system constants.

Note that `SwSystemconstDependentFormula.sysc` reflects the internal value of the `SwSystemconst`.

Note that `SwSystemconstDependentFormula.syscString` reflects the string representation value of the SwSystemconst. This is in particular the symbol for representation of a CompuScale in C determined according to [constr_1145] in [15].

The formula language is described in detail in Section 4.8. In this section, we concentrate on the variant handling related classes that are derived from `SwSystemconstDependentFormula`, namely `AttributeValueVariationPoint` and `ConditionByFormula`:

ConditionByFormula is aggregated by VariationPoint and decides whether the element to which the VariationPoint is attached actually exists. This is used in all patterns except the *attribute value pattern*.

The return value of this formula is always interpreted as a boolean: 0 equals *false*, any other value is interpreted as *true*.

AttributeValueVariationPoint is primarily used to provide values in the *attribute value pattern* (Section 7.4). Since the *attribute value pattern* is implicitly used to define the condition of a *PostBuild* variation point, it may also be used in every other pattern.

AttributeValueVariationPoint further splits into six subclasses, namely NumericalValueVariationPoint, FloatValueVariationPoint, IntegerValueVariationPoint, BooleanValueVariationPoint, PositiveIntegerValueVariationPoint and UnlimitedIntegerValueVariationPoint.

These subclasses provide information on the expected return type of the formula (see Chapter 4.8.2.2, and correspond to the AUTOSAR primitive types Numerical, Float, Integer, Boolean, PositiveInteger and NumericalInteger.

[constr_2516] Return type of Formula ⌈ When such a formula is evaluated by a software tool, and the return value of the formula is must be compatible to the type of the attribute in the pure meta-model. ⌋

Class	≪**atpMixedString**≫ **SwSystemconstDependentFormula (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents an expression depending on system constants.			
Base	ARObject,FormulaExpression			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
sysc	SwSystemconst	1	ref	This refers to a system constant. The internal (coded) value of the system constant shall be used. **Tags:** xml.sequenceOffset=50
syscString	SwSystemconst	1	ref	syscString indicates that the referenced systm constant shall be evaluated as a string according to Constr_1145 (in SWCT).

Table 7.10: SwSystemconstDependentFormula

Class	≪`atpMixedString`≫ **ConditionByFormula**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents a condition which is computed based on system constants according to the specified expression. The expected result is a considered as boolean value. The result of the expression is interpreted as a condition. • "0" represents "false"; • a value greater than zero is considered "true"			
Base	ARObject,FormulaExpression,SwSystemconstDependentFormula			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
bindingTime	BindingTimeEnum	1	attr	This attribute specifies the point in time when condition may be evaluated at earliest. At this point in time all referenced system constants must have a value. **Tags:** xml.attribute=true

Table 7.11: ConditionByFormula

Class	≪`atpMixedString`≫ **AttributeValueVariationPoint (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents the ability to derive the value of the Attribute from a system constant (by SwSystemconstDependentFormula). It also provides a bindingTime.			
Base	ARObject,FormulaExpression,SwSystemconstDependentFormula			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
bindingTime	BindingTimeEnum	0..1	attr	This is the binding time in which the attribute value needs to be bound. If this attribute is missing, the attribute is not a variation point. In particular this means that It needs to be a single value according to the type specified in the pure model. It is an error if it is still a formula. **Tags:** xml.attribute=true
blueprintValue	String	0..1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:** xml.attribute=true
sd	String	0..1	attr	This special data is provided to allow synchronisation of Attribute value variation points with variant management systems. The usage is subject of agreement between the involved parties. **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
shortLabel	Identifier	0..1	ref	This allows to identify the variation point. It is also intended to allow RTE support for CompileTime Variation points. **Tags:** xml.attribute=true

Table 7.12: AttributeValueVariationPoint

Class	≪`atpMixedString`≫ **NumericalValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents an attribute value variation point for Numerical attributes. Note that this class might be used in the extended meta-model only.			
Base	ARObject,AbstractNumericalVariationPoint,AttributeValueVariationPoint,Formula Expression,SwSystemconstDependentFormula			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 7.13: NumericalValueVariationPoint

Class	≪`atpMixedString`≫ **FloatValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents an attribute value variation point for Float attributes. Note that this class might be used in the extended meta-model only			
Base	ARObject,AttributeValueVariationPoint,FormulaExpression,SwSystemconst DependentFormula			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 7.14: FloatValueVariationPoint

Class	≪`atpMixedString`≫ **IntegerValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents an attribute value variation point for Integer attributes. Note that this class might be used in the extended meta-model only.			
Base	ARObject,AttributeValueVariationPoint,FormulaExpression,SwSystemconst DependentFormula			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 7.15: IntegerValueVariationPoint

Class	≪`atpMixedString`≫ **BooleanValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents an attribute value variation point for Boolean attributes. Note that this class might be used in the extended meta-model on			
Base	ARObject,AttributeValueVariationPoint,FormulaExpression,SwSystemconst DependentFormula			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 7.16: BooleanValueVariationPoint

Class	≪`atpMixedString`≫ **PositiveIntegerValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents an attribute value variation point for positive Integer attributes. Note that this class might be used in the extended meta-model only.			
Base	ARObject,AttributeValueVariationPoint,FormulaExpression,SwSystemconst DependentFormula			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 7.17: PositiveIntegerValueVariationPoint

Class	≪`atpMixedString`≫ **UnlimitedIntegerValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents an attribute value variation point for unlimited Integer attributes. Note that this class might be used in the extended meta-model only.			
Base	ARObject,AttributeValueVariationPoint,FormulaExpression,SwSystemconst DependentFormula			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 7.18: UnlimitedIntegerValueVariationPoint

7.6.9 Combining *PreBuild* and *PostBuild* Variation Points

If `Vh.latestBindingTime` is set to `PostBuild` ([constr_2517]) for a particular variation point, then this variation point may have a *PostBuild* branch represented by `PostBuildVariationCondition`[16]. However, it may also contain a *PreBuild* branch.

This is because the *PreBuild* and *PostBuild* branches of `VariationPoint` are not mutually exclusive. It is possible to define a variation point as *both PreBuild and Post-*

[16]Vice versa, whether that a variation point is *PostBuild* can be recognized from the fact that it contains `PostBuildVariationCondition`.

Build. If the PreBuild condition is false, it is not expected that the variant object (including the PostBuild condition) will be implemented.

In other words, a system constant expression in a variation point may be used to disable the *PostBuild* variability even at *PreBuild* time.

Table 7.19 summarizes the options.

PreBuild	PostBuild	
No Condition at all	No Condition at all	The element to which the VP is attached is always selected
No Condition at all	Unbound Condition	Pure PostBuild Variation Point
No Condition at all	Condition bound to true	Bound, selected PostBuild Variation Point (not visible in XML)
No Condition at all	Condition bound to false	Bound, deselected PostBuild Variation Point (not visible in XML)
Unbound Condition	No Condition at all	Pure PreBuild Variation Point
Unbound Condition	Unbound Condition	PreBuild selectable Postbuild Variation Point
Unbound Condition	Condition bound to true	PreBuild selectable Postbuild Variation Point (not visible in XML)
Unbound Condition	Condition bound to false	PreBuild selectable Postbuild Variation Point (not visible in XML)
Condition bound to true	No Condition at all	The element to which the VP is attached is always selected
Condition bound to true	Unbound Condition	Pure PostBuild Variation Point
Condition bound to true	Condition bound to true	Bound, selected PostBuild Variation Point (not visible in XML)
Condition bound to true	Condition bound to false	Bound, deselected PostBuild Variation Point (not visible in XML)
Condition bound to false	No Condition at all	Deselected PreBuild Variation Point, no PostBuild Variation Point
Condition bound to false	Unbound Condition	Deselected PreBuild Variation Point, no PostBuild Variation Point
Condition bound to false	Condition bound to true	Deselected PreBuild Variation Point, no PostBuild Variation Point
Condition bound to false	Condition bound to false	Deselected PreBuild Variation Point, no PostBuild Variation Point

Table 7.19: Combining *PreBuild* and *PostBuild* Variation Points

7.6.10 Notes and Restrictions

1. It is not supported to aggregate more than one `VariationPoint` at the same location. It is however possible to define both *PreBuild* and *PostBuild* conditions for a single variation point.
2. It is not possible to state multiple binding times for a single variation point. The rationale for restriction is that if multiple binding times would really be used at the same location, then their conditions are likely to differ anyway. That is, there would not be a single variation point with multiple binding times, but several variation points instead.

3. Due to the very nature of dealing with variants, it is possible to have multiple elements with the *same* `ShortName` until all *PreBuild* variants are resolved.

   This also means that the checking the model for consistency might not be fully possible until after all variants are resolved. This is because one purpose of variant handling is to model several incompatible variants and provide means to select one of the and discard the others, but this implies that the model cannot be fully consistent until this selection has been made.

   See also Section 7.6.2 for more details on this issue.

4. The reason for handling *PreBuild* and *PostBuild* variation points differently is that if we would use only a `ConditionByFormula` (i.e., only the *PreBuild* branch) in both cases, then this condition would have to be evaluated at startup time. However, this would impose a performance penalty that is generally not acceptable. Hence, we use a simpler approach for *PostBuild* variation points which requires only a comparison.

5. References within an AUTOSAR Model may also be from invariant (respectively *PreBuild* variant) elements to post build variant elements if all variants of the variant element do have a meaning for the invariant elements.

   An sample use case is conditionaly existing `ComponentPrototype`, with `LatesBindingTime` = `PostBuild`. The runnable to task mapping of the variant `ComponentPrototype` is a `PreCompileTime` ECUC parameter. But this mapping has a meaning for all variants of the variant `ComponentPrototype`, because it is resolved before and does not need any additional condition.

6. A `shortLabel` could also be implemented by making a `VariationPoint` an `Identifiable` (see Section4.4). However, `Identifiable` would be expensive for our purposes:

   - `Identifiable` has a significantly higher XML footprint than the `shortLabel` attribute.
   - A `shortLabel` is always optional while `Identifiable` adds a required `shortName`.

   In addition, `shortLabel` is intended for local identification (within the next enclosing `Identifiable`), while `Identifiable` is intended for reference purposes.

7.6.11 Using Variation Points for Blueprinting

As specified in [9], `VariationPoint` and `AttibuteValueVariationPoint` are also used to specify details of deriving objects from blueprints.

Variation handling in Blueprints works differently from AUTOSAR variant handling elsewhere:

- Processing Blueprints can be seen as a separate binding time that occurs before SystemDesignTime.
- The model does not give precise instructions how to handle the variation. Instead, only a textual description of what needs to be done is available.
- Variation points may occur for all elements in the blueprint that are subclasses of ARElement; [constr_2537] states that it does not apply for Blueprints.

7.6.11.1 When is a variation point a Blueprint variation point?

The class VariationPoint (see Figure 7.8) aggregates a DocumentationBlock in the role blueprintCondition. This object may only be aggregated in blueprints – that is, it may only be present if the VariationPoint lives in an AUTOSAR package of category BLUEPRINT. It may not be present In a system configuration.

If such an DocumentationBlock is present, then it contains instructions how to further process this variation point. The specific format of these instructions is not prescribed in detail, as the instructions are meant for humans or specialized (probably proprietary) tools.

Similarly, AttributeValueVariationPoint has an attribute blueprintValue that serves the same purpose as a DocumentationBlock.

A further consequence of the fact that DocumentationBlock is only allowed in a blueprint is documented in [constr_2559]. In plain text, this constraint says that there cannot be variation points within a DocumentationBlock outside of blueprints. The rationale for this is that such variations would have to be resolved at SystemDesignTime or later, which comes *after* the blueprint has been processed.

7.6.11.2 BluePrintDerivationTime

To support blueprints, the tag Vh.latestBindingTime may have the value BluePrintDerivationTime. Such a variation point may only be present in a blueprint and may *not* be copied to a system configuration.

In this case, a variation point cannot have a swSysCond nor postbuildVariantCondition (as defined by [constr_2558] in [9]) because the information contained in these fields cannot be processed at BluePrintDerivationTime.

Similarly, the value an AttributeValueVariationPoint has no meaning in this case. Therefore it shall not be evaluated and the value shall be undefined. See also Section 7.4.2.

7.6.11.3 Which AUTOSAR model elements can be blueprint variation points?

Non-blueprint variation points – that is, variation points which are resolved at `SystemDesignTime` or later – may not be used everywhere in a model. Their applicability is restricted by the metamodel to those locations that carry the stereotype ≪`atpVariation`≫. There are good reasons to do this; for example, the RTE must be able to cope with the variation point at this location and must be able to generate appropriate code.

The AUTOSAR elements `PackableElement` and `ARElement` – which derives from it – form a special case. Since `PackableElement` is variable with a latest binding time of `SystemDesignTime`, *any* `ARElement` could be variable, which is clearly not intended. Hence, [constr_2537] restricts that to certain elements.

In Blueprints, [constr_2537] has been relaxed, and any `ARElement` may be a variation point.

There is however a caveat: when elements are copied from a variation point to a system configuration, then only those variation points may be transferred that are allowed to be variation points at `SystemDesignTime` or later. Hence, any variation of an `ARElement` (or something derived from it) in a blueprint that would be prohibited by [constr_2537] (which limits variation points to certain elements) has an implicit latest binding time of `BluePrintDerivationTime`.

7.7 Evaluated Variants

7.7.1 Motivation

Variant handling does not end with a description of *where* variation occurs (that is, the patterns we described in the previous sections of this chapter). Quite often, this description implies a huge number of variants[17], but only a subset of those is actually used.

This may be because the software is built to support a wider range of options than those of one particular OEM. But since the supplier has several OEM's as customer, he might design the software in such a way that it satisfies all variants. What is shipped to the OEM may only contain artifacts for his particular variants[18], or is at least certified for only those.

The variations are described by sets of system constant values. Hence, there is a need to describe which combinations of system constant values are valid. This provides the basis for OEMs and suppliers to exchange information on this subject in a standardized way.

[17] Just five alternatives with three mutually exclusive options each yield a total of $3^5 = 243$ options.

[18] Especially if the variability is at `SystemDesignTime` or `PreCompileTime`; although this is generally not possible for *PostBuild* variations.

7.7.2 Example

“APPROVED”	Basic	**Economy**	**Senior**	Sportive	Junior
Turbo	0	**0**	**1**	1	0
Gear	0	**1**	**0**	1	0
Light	0	**1**	**2**	3	0
Sunroof	0	**0**	**0**	1	1

Table 7.20: Evaluated Variant Example, full table

Table 7.20 illustrates a example where we have four system constants (*Turbo*, *Gear*, *Light* and *Sunroof*) which can assume integer values. In this example five variants were evaluated and named *Basic*, *Economy*, *Senior*, *Sportive*, *Junior*.

Basic, *Economy*, *Senior*, *Sportive*, *Junior* are called **predefined variants**. Each predefined variant is a combination of system constant values[19]. In other words, a `PredefinedVariant` is a column in the table above representing all evaluated variants.

The result of the evaluation is stated by the attribute `approvalStatus` in `EvaluatedVariantSet`. Corresponding to the example above the bold columns can be represented by an `EvaluatedVariantSet` with `approvalStatus` set to “APPROVED” as shown in table 7.21.

Furthermore, let us assume that a supplier is able to provide all five combinations (internally), but a fictitious OEM is interested in buying only *Economy* and *Senior*, as indicated by the bold column in Table 7.20.

In this case, the table that is exchanged between OEM and supplier will contain only the two columns (predefined variants) *Economy* and *Senior*.

”APPROVED”	**Economy**	**Senior**
Turbo	0	1
Gear	1	0
Light	1	2
Sunroof	0	0

Table 7.21: Table that is exchanged between supplier and OEM

Table 7.21 shows the table that is exchanged between OEM and supplier. This table is also the basis for the XML example in Section 7.7.5.

7.7.2.1 Beyond the example

There are four more aspects in the concept for *evaluated variants* which are not shown in the above example:

[19] A predefined variant also includes values for *PostBuild* variation points, which are omitted in Table 7.20 for clarity.

1. An evaluated variant may refer to a specific component (or other element) for which the approval status in table applies.
2. A predefined variant may not only define values for system constants, but also for PostBuild variant criteria.
3. Columns in the table – even sub-columns – may be re-used by other tables. This is done by implementing the table through references, not aggregations. Both features are helpful if the data for a table of evaluated variants gets reused, or comes from different sources.
4. An evaluated variant may have an approval status which further details the meaning of the table. An evaluated variant may be “approved’, in which case the table contains predefined variants that are known to work, or it may be “rejected”, in which cases the predefined variants are known *not* to work.
5. A `PredefinedVariant` can include other `PredefinedVariant`s with a `includedVariant` association.

7.7.2.2 Use Cases covered in the example

The example above covers the following use cases:

- An integrator can use a table of evaluated variants to check whether a certain non-variant system (i.e. one where the variants have been resolved) is based on a predefined variant.
- A system designer can import preconfigured settings to build a particular variant.
- A component provider can use this mechanism to deliver a set of valid variants to a user of a component. This does not need to be the whole set of valid variants; for example the user may only be entitled to see (i.e., get information on) a certain subset.

7.7.3 Description

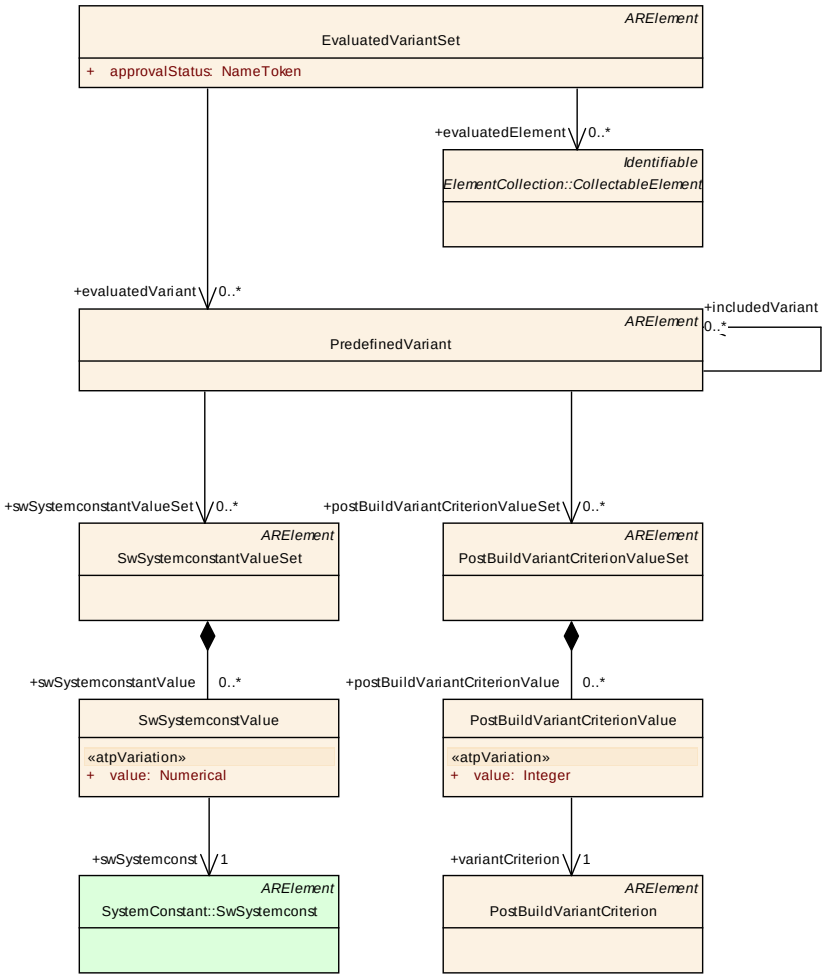

Figure 7.13: `EvaluatedVariantSet`

Tables 7.20 and 7.21 translate to Figure 7.13 as follows:

`EvaluatedVariantSet` The whole table is represented by the class `EvaluatedVariantSet`. `EvaluatedVariantSet` is an `ARElement`, so it has its own `shortName`, which is the name of the table.

It is possible to have multiple `EvaluatedVariantSet`s. If there are several such sets, each set establishes a validity. For example, unit tests may use their own specialized `EvaluatedVariantSet`s. Individual sets may be addressed by `shortName`.

`PredefinedVariant` An `EvaluatedVariantSet` contains a number of `PredefinedVariant`s. Each `PredefinedVariant` plus its included variants are a column in the table. The name of the column is the `shortName` of the `PredefinedVariant`.

`SwSystemconstantValueSet` A `PredefinedVariant` contains a number of `SwSystemconstantValueSet` objects. In the simplest case, there is only one

such object, which represents the entries of the column. More precisely, the PredefinedVariant represents the column including the header, while the SwSystemconstantValueSet is all that is below the header.

It is also possible to distribute the entries of a column over several SwSystemconstantValueSet objects. To remain with the picture that was drawn in Figure 7.20, each column is then composed of several SwSystemconstantValueSets, whose contents are concatenated.

[constr_2519] PredefinedVariants need to be consistent ⌈ If a PredefinedVariant plus its includedVariants references more than one SwSystemconstantValueSet all value attributes in SwSystemconstValues for a particular SwSystemconst must be identical. ⌋

The reason behind using several SwSystemconstantValueSets is to allow a predefined variant to be composed of system constant assignments that come from different sources.

By constraint [constr_2519] contradicting value assignment are positively avoided.

SwSystemconstantValue A SwSystemconstantValueSet contains a number of SwSystemconstantValue objects, each of which represents a cell in the table, and implements – as the name says – a value for a single system constant.

The value that is stored in the cell is represented by the attribute value, which in turn subject to variation through *attribute value pattern* (see Section 7.4).

The primary motivation for using a variation point here is convenience: the value is determined by an expression[20], and this is exactly what an attribute value variation point does.

SwSystemconst Each SwSystemconstValue provides a reference to a SwSystemconst. This is the system constant whose value is defined by SwSystemconstValue.

SwSystemconstantValueSet, SwSystemconstantValue and SwSystemconst define *prebuild* variants. There is a second branch for postbuild variants:

PostBuildVariantCriterionValueSet is the *postbuild* analogue for SwSystemconstantValueSet.

PostBuildVariantCriterionValue is the *postbuild* analogue for SwSystemconstValue.

PostBuildVariantCriterion is the *postbuild* analogue for SwSystemconst.

When both *prebuild* and *postbuild* variants are defined, then the *postbuild* variants are valid for *all prebuild* variants.

[20] In practice, the expression should consist of a single system constant, most of the time.

Furthermore, PredefinedVariant, SwSystemconstantValueSet and PostBuildVariantCriterionValueSet are referenced, rather than aggregated, to enable reuse of variants. For example, a vendor might have several PredefinedVariant collections – one for each OEM – and reuse them in separate EvaluatedVariantSets.

7.7.3.1 evaluatedElement

A EvaluatedVariantSet provides a reference to one or more CollectableElements (see chapter 4.3). This is used to identify the packages and elements that are covered by the EvaluatedVariantSet. Note that EvaluatedVariantSet is also CollectableElement.

[constr_2507] EvaluatedVariantSet must not refer to itself ⌈ An EvaluatedVariantSet must not refer to itself directly or via other EvaluatedVariantSet. ⌋

For more details refer to Section 7.7.4.

7.7.3.2 approvalStatus

The attribute approvalStatus of an EvaluatedVariantSet further details the status of the evaluated variant:

1. APPROVED An “approved” variant is known to work.
2. REJECTED An “rejected” variant is known *not* to work.

Note that all other values are use case specific. Hence EvaluatedVariantSet with approvalStatus other than APPROVED or REJECTED is not defined by the standard and therefore shall be ignored when evaluating a system according to chapter 7.7.4.

7.7.3.3 includedVariant

The association includedVariant defines that settings of the referenced PredefinedVariants are handled as part of the settings of the referencing PredefinedVariant.

Suppose a variant rich system is composed out of variant rich sub systems delivered by several parties. In this case the providers of the subsystems need to define appropriate EvaluatedVariantSet for their delivery. Additionally the responsible party for the system needs to specify EvaluatedVariantSet for the entire system.

In order to do this he can use the includedVariants to refer to the definition of the PredefinedVariants of the sub system. Without this he would need to repeat those definitions. This would require a knowledge about the sub system and reduce

the maintainablility of the system. By using includedVariants the creator of the system does not need any knowledge about the SwSystemconstantValueSets and PostBuildVariantCriterionValueSet used in the sub system.

7.7.4 Consistency

A particular EvaluatedVariantSet refers to CollectableElements in order to express their approval status. As EvaluatedVariantSet is also a CollectableElement, a hierarchy of EvaluatedVariantSets can occur.

On the other hand the meta model establishes another hierarchy by aggregation of objects. It is important to clearly distinguish these two hierarchies when considering evalutated variants.

This section defines the details regarding consistency of such hierarchies.

Generally the status REJECTED takes precedence over the status APPROVED. That is, if e.g. an “approved” package contains a “rejected” software component, the whole package shall be regarded as REJECTED for this EvalulatedVariantSet.

A CollectableElement (see chapter 3) is *rejected* for a given variant if

- it is referenced by at least one appropriate EvaluatedVariantSet with approvalStatus set to *REJECTED*
- or aggregates (possibly over many levels) a CollectableElement which is referenced in an appropriate EvaluatedVariantSet with approvalStatus set to *REJECTED*.

A CollectableElement (see chapter 3) is *approved* for a given variant if

- it is referenced by at least one appropriate EvaluatedVariantSet with approvalStatus set to *APPROVED* or is not a *rejected* EvaluatedVariantSet[21].
- and is not a *rejected* CollectableElement.

7.7.5 XML Example for EvaluatedVariantSet

The following listing illustrates how an EvaluatedVariantSet is expressd in XML:

Listing 7.1: Example for evaluated variant in ARXML

```
<AR-PACKAGE>
  <SHORT-NAME>Variants</SHORT-NAME>
  <ELEMENTS>
    <PREDEFINED-VARIANT>
      <SHORT-NAME>Basic</SHORT-NAME>
      <SW-SYSTEMCONSTANT-VALUE-SET-REFS>
```

[21] This makes sure that EvaluatedVariantSet are considered approved by default.

```
          <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST="SW-SYSTEMCONSTANT-VALUE-
              SET">/SystemConstantValues/V1</SW-SYSTEMCONSTANT-VALUE-SET-
              REF>
          <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST="SW-SYSTEMCONSTANT-VALUE-
              SET">/SystemConstantValues/V2</SW-SYSTEMCONSTANT-VALUE-SET-
              REF>
        </SW-SYSTEMCONSTANT-VALUE-SET-REFS>
      </PREDEFINED-VARIANT>
      <PREDEFINED-VARIANT>
        <SHORT-NAME>Economy</SHORT-NAME>
        <SW-SYSTEMCONSTANT-VALUE-SET-REFS>
          <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST="SW-SYSTEMCONSTANT-VALUE-
              SET">/SystemConstantValues/V1</SW-SYSTEMCONSTANT-VALUE-SET-
              REF>
          <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST="SW-SYSTEMCONSTANT-VALUE-
              SET">/SystemConstantValues/V3</SW-SYSTEMCONSTANT-VALUE-SET-
              REF>
        </SW-SYSTEMCONSTANT-VALUE-SET-REFS>
      </PREDEFINED-VARIANT>
      <PREDEFINED-VARIANT>
        <SHORT-NAME>Senior</SHORT-NAME>
        <SW-SYSTEMCONSTANT-VALUE-SET-REFS>
          <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST="SW-SYSTEMCONSTANT-VALUE-
              SET">/SystemConstantValues/V4</SW-SYSTEMCONSTANT-VALUE-SET-
              REF>
        </SW-SYSTEMCONSTANT-VALUE-SET-REFS>
      </PREDEFINED-VARIANT>
      <PREDEFINED-VARIANT>
        <SHORT-NAME>Sportive</SHORT-NAME>
        <SW-SYSTEMCONSTANT-VALUE-SET-REFS>
          <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST="SW-SYSTEMCONSTANT-VALUE-
              SET">/SystemConstantValues/V5</SW-SYSTEMCONSTANT-VALUE-SET-
              REF>
        </SW-SYSTEMCONSTANT-VALUE-SET-REFS>
      </PREDEFINED-VARIANT>
    </ELEMENTS>
  </AR-PACKAGE>

  <!--  now we have the systemconstant value sets -->
  <AR-PACKAGE>
    <SHORT-NAME>SystemConstantValues</SHORT-NAME>
    <ELEMENTS>
      <SW-SYSTEMCONSTANT-VALUE-SET>
        <SHORT-NAME>V1</SHORT-NAME>
        <SW-SYSTEMCONSTANT-VALUES>
          <SW-SYSTEMCONST-VALUE>
            <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Turbo</SW-
                SYSTEMCONST-REF>
            <VALUE>0</VALUE>
          </SW-SYSTEMCONST-VALUE>
          <SW-SYSTEMCONST-VALUE>
            <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Sunroof</SW-
                SYSTEMCONST-REF>
            <VALUE>0</VALUE>
          </SW-SYSTEMCONST-VALUE>
        </SW-SYSTEMCONSTANT-VALUES>
```

```
</SW-SYSTEMCONSTANT-VALUE-SET>
<SW-SYSTEMCONSTANT-VALUE-SET>
  <SHORT-NAME>V2</SHORT-NAME>
  <SW-SYSTEMCONSTANT-VALUES>
    <SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Gear</SW-
          SYSTEMCONST-REF>
      <VALUE>0</VALUE>
    </SW-SYSTEMCONST-VALUE>
    <SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Light</SW-
          SYSTEMCONST-REF>
      <VALUE>0</VALUE>
    </SW-SYSTEMCONST-VALUE>
  </SW-SYSTEMCONSTANT-VALUES>
</SW-SYSTEMCONSTANT-VALUE-SET>
<SW-SYSTEMCONSTANT-VALUE-SET>
  <SHORT-NAME>V3</SHORT-NAME>
  <SW-SYSTEMCONSTANT-VALUES>
    <SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Gear</SW-
          SYSTEMCONST-REF>
      <VALUE>1</VALUE>
    </SW-SYSTEMCONST-VALUE>
    <SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Light</SW-
          SYSTEMCONST-REF>
      <VALUE>1</VALUE>
    </SW-SYSTEMCONST-VALUE>
  </SW-SYSTEMCONSTANT-VALUES>
</SW-SYSTEMCONSTANT-VALUE-SET>
<!-- note that this set is used in all variants above -->
<SW-SYSTEMCONSTANT-VALUE-SET>
  <SHORT-NAME>v4</SHORT-NAME>
  <SW-SYSTEMCONSTANT-VALUES>
    <SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Turbo</SW-
          SYSTEMCONST-REF>
      <VALUE>1</VALUE>
    </SW-SYSTEMCONST-VALUE>
    <SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Gear</SW-
          SYSTEMCONST-REF>
      <VALUE>0</VALUE>
    </SW-SYSTEMCONST-VALUE>
    <SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Light</SW-
          SYSTEMCONST-REF>
      <VALUE>2</VALUE>
    </SW-SYSTEMCONST-VALUE>
    <SW-SYSTEMCONST-VALUE>
      <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Sunroof</SW-
          SYSTEMCONST-REF>
      <VALUE>0</VALUE>
    </SW-SYSTEMCONST-VALUE>
  </SW-SYSTEMCONSTANT-VALUES>
```

```
      </SW-SYSTEMCONSTANT-VALUE-SET>
      <SW-SYSTEMCONSTANT-VALUE-SET>
        <SHORT-NAME>v5</SHORT-NAME>
        <SW-SYSTEMCONSTANT-VALUES>
          <SW-SYSTEMCONST-VALUE>
            <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Turbo</SW-
               SYSTEMCONST-REF>
            <VALUE>1</VALUE>
          </SW-SYSTEMCONST-VALUE>
          <SW-SYSTEMCONST-VALUE>
            <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Gear</SW-
               SYSTEMCONST-REF>
            <VALUE>1</VALUE>
          </SW-SYSTEMCONST-VALUE>
          <SW-SYSTEMCONST-VALUE>
            <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Light</SW-
               SYSTEMCONST-REF>
            <VALUE>3</VALUE>
          </SW-SYSTEMCONST-VALUE>
          <SW-SYSTEMCONST-VALUE>
            <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">Sunroof</SW-
               SYSTEMCONST-REF>
            <VALUE>1</VALUE>
          </SW-SYSTEMCONST-VALUE>
        </SW-SYSTEMCONSTANT-VALUES>
      </SW-SYSTEMCONSTANT-VALUE-SET>
    </ELEMENTS>
  </AR-PACKAGE>


  <!--  here we have the evaluated variants -->
  <AR-PACKAGE>
    <SHORT-NAME>EvaluatedVariants</SHORT-NAME>
    <ELEMENTS>
      <EVALUATED-VARIANT-SET>
        <SHORT-NAME>foobar</SHORT-NAME>
        <EVALUATED-ELEMENT-REFS>
          <EVALUATED-ELEMENT-REF DEST="APPLICATION-SW-COMPONENT-TYPE">/
             Components/foo</EVALUATED-ELEMENT-REF>
          <EVALUATED-ELEMENT-REF DEST="APPLICATION-SW-COMPONENT-TYPE">/
             Components/bar</EVALUATED-ELEMENT-REF>
        </EVALUATED-ELEMENT-REFS>
        <EVALUATED-VARIANT-REFS>
          <EVALUATED-VARIANT-REF DEST="PREDEFINED-VARIANT">/Variants/
             Economy</EVALUATED-VARIANT-REF>
          <EVALUATED-VARIANT-REF DEST="PREDEFINED-VARIANT">/Variants/
             Senior</EVALUATED-VARIANT-REF>
        </EVALUATED-VARIANT-REFS>
      </EVALUATED-VARIANT-SET>
    </ELEMENTS>
  </AR-PACKAGE>


  <!-- here we have the components -->
  <AR-PACKAGE>
    <SHORT-NAME>Components</SHORT-NAME>
```

```
      <ELEMENTS>
        <APPLICATION-SW-COMPONENT-TYPE>
          <SHORT-NAME>foo</SHORT-NAME>
        </APPLICATION-SW-COMPONENT-TYPE>
        <APPLICATION-SW-COMPONENT-TYPE>
          <SHORT-NAME>bar</SHORT-NAME>
        </APPLICATION-SW-COMPONENT-TYPE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
```

Note: system constants are not only identified by their name (as implied in Table 7.20), but they are in fact identified by the full reference. This avoids name clashes in hierarchies, e.g. if two (sub)components come from different vendors that for some reason have used the same names.

7.7.6 Classtables

Class	**EvaluatedVariantSet**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta class represents the ability to express if a set of ArElements is able to support one or more particular variants. In other words, for a given set of evaluatedElements this meta class represents a table of evaluated variants, where each PredefinedVariant represents one column. In this colum each descendant swSystemconstantValue resp. postbuildVariantCriterionValue represents one entry. In a graphical representation each swSystemconstantValueSet / postBuildVariantCriterionValueSet could be used as an intermediate headline in the table column. If the approvalStatus is "APPROVED" it expresses that the collection of CollectableElements is known be valid for the given evaluatedVariants. Note that the EvaluatedVariantSet is a CollectableElement. This allows to establish a hierarchy of EvaluatedVariantSets. **Tags:** atp.recommendedPackage=EvaluatedVariantSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	Datatype	Mul.	Kind	Note
approvalStatus	NameToken	1	attr	Defines the approval status of a predefined variant. Two values are predefined: "APPROVED" and "REJECTED": • Approved variants are known to work. • Rejected variants are known NOT to work. Further values can be approved on a per-company basis; within AUTOSAR only "APPROVED" and "REJECTED" should be recognized.
evaluatedElement	CollectableElement	*	ref	This represents a particular element which is evaluated in context of the EvaluatedVariants. The approvalStatus applies to this element (and all of its descendants). In other words, the referenced elements are those that were considered when the predefined variant was evaluated.
evaluatedVariant	PredefinedVariant	*	ref	This metaclass represents one particular variant which was evaluated. LowerMultiplicity is set to 0 to support a stepwise approach.

Table 7.22: EvaluatedVariantSet

Class	PredefinedVariant			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This specifies one predefined variant. It is characterized by the union of all system constant values and post-build variant criterion values aggregated within all referenced system constant value sets and post build variant criterion value sets plus the value sets of the included variants. **Tags:** atp.recommendedPackage=PredefinedVariants			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
includedVariant	PredefinedVariant	*	ref	The associated variants are considered part of this PredefinedVariant. This means the settings of the included variants are included in the settings of the referencing PredefinedVariant. Nevertheless the included variants might be included in several predefined variants.
postBuildVariantCriterionValueSet	PostBuildVariantCriterionValueSet	*	ref	This is the postBuildVariantCriterionValueSet contributing to the predefinded variant.
swSystemconstantValueSet	SwSystemconstantValueSet	*	ref	This ist the set of Systemconstant Values contributing to the predefined variant.

Table 7.23: PredefinedVariant

Class	**SwSystemconstantValueSet**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to specify a set of system constant values. **Tags:** atp.recommendedPackage=SwSystemconstantValueSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
swSystemconstantValue	SwSystemconstValue	*	aggr	This is one particular value of a system constant.

Table 7.24: SwSystemconstantValueSet

Class	**SwSystemconstValue**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class assigns a particular value to a system constant.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
annotation	Annotation	*	aggr	This provides the ability to add information why the value is set like it is. **Tags:** xml.sequenceOffset=30
swSystemconst	SwSystemconst	1	ref	This is the system constant to which the value applies. **Tags:** xml.sequenceOffset=10
value	Numerical	1	attr	This is the particular value of a system constant. It is specified as Numerical. Further restrictions may apply by the definition of the system constant. The value attribute defines the internal value of the SwSystemconst as it is processed in the Formula Language. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=20

Table 7.25: SwSystemconstValue

Class	**SwSystemconst**			
Package	M2::AUTOSARTemplates::CommonStructure::SystemConstant			
Note	This element defines a system constant which serves an input to select a particular variation point. In particular a system constant serves as an operand of the binding function (swSyscond) in a Variation point. Note that the binding process can only happen if a value was assigned to to the referenced system constants. **Tags:** atp.recommendedPackage=SwSystemconsts			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
swDataDefProps	SwDataDefProps	0..1	aggr	This denotes the data defintion properties of the system constant. In particular it is the limits and - in case the system constant is an enumeration - the compu method. **Tags:** xml.sequenceOffset=40

Table 7.26: SwSystemconst

Class	**PostBuildVariantCriterionValueSet**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to denote one set of postBuildVariantCriterionValues. **Tags:** atp.recommendedPackage=PostBuildVariantCriterionValueSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
postBuildVariantCriterionValue	PostBuildVariantCriterionValue	*	aggr	This is is one particular postbuild variant criterion/value pair being part of the PostBuildVariantSet.

Table 7.27: PostBuildVariantCriterionValueSet

Class	**PostBuildVariantCriterionValue**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies a the value which must be assigned to a particular variant criterion in order to bind the variation point. If multiple criterion/value pairs are specified, they all must must match to bind the variation point.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
annotation	Annotation	*	aggr	This provides the ability to add information why the value is set like it is. **Tags:** xml.sequenceOffset=30

Attribute	Datatype	Mul.	Kind	Note
value	Integer	1	attr	This is the particular value of the post-build variant criterion. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=20
variantCriterion	PostBuildVariantCriterion	1	ref	This association selects the variant criterion whose value is specified. **Tags:** xml.sequenceOffset=10

Table 7.28: PostBuildVariantCriterionValue

Class	PostBuildVariantCriterion			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies one particular PostBuildVariantSelector. **Tags:** atp.recommendedPackage=PostBuildVariantCriterions			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
compuMethod	CompuMethod	1	ref	The compuMethod specifies the possible values for the variant criterion serving as an enumerator.

Table 7.29: PostBuildVariantCriterion

7.8 Choosing Variants

In this section, we describe how to represent a particular variant of an AUTOSAR system.

7.8.1 What is a Variant?

A variant is a collection of bound variation points. In other words, it is represented by a complete assignment of values to system constants.

More formally, a *variant* is characterized by the following:

1. **Binding Time.** Variants are tied to a particular binding time. This binding time is given in the following model elements:
   - `ConditionByFormula`, which is aggregated by `VariationPoint` in the *aggregation pattern* (Figure 7.2), the *association pattern* (Figure 7.3), and the the *property set pattern* (Figures 7.7, 7.6).
   - `AttributeValueVariationPoint` in the *attribute value pattern* (Figure 7.4).

[constr_2518] Binding time is constrained ⌈ Note that this binding time is again constrained by the value of the tag `Vh.latestBindingTime`. ⌋

See Section 7.1.6 for details.

2. **Variation Points.** The set of variation points that need to bound at the binding time of the variant. In the model, this affects the following elements:
   - `VariationPoint`. This is used in the *aggregation pattern*, (Figure 7.2), the *association pattern* (Figure 7.3) and the *property set pattern* (Figures 7.7, 7.6).
   - `AttributeValueVariationPoint` in the *attribute value pattern* (Figure 7.4).
3. **System Constants.** The set of system constants that are referenced from within variation points. System constants are referenced from `SwSystemconstDependentFormula` and its subclasses (Section 7.6.8).
4. **Value assignments.** The system constants determined in the previous step need to be assigned values.

7.8.2 Valid Variants

A *valid variant* is defined as follows:

1. A valid variant $\mathcal{V}$ for a binding time $\mathcal{T}$ is an assignment of values to all system constants that are referenced by all variation points which have the binding time $\mathcal{T}$.
2. There exists no `PredefinedVariant` that covers the systems in $\mathcal{V}$, and its status is `REJECTED`.

7.9 Examples

In this section, we provide four examples for variant handling, each focusing on a single template. We also provide possible implementations to illustrate how these use cases might be translated into practice. It should however be noted that these are just sample implementations, and are not necessarily the only possible way to realize a particular use case.

7.9.1 Example for *Aggregation Pattern*

Description Software component type with variant ports.

Use Case A body control application uses one of two light controllers: low end systems uses the *low-end-light control*, while a high end systems uses *adaptive-curve light-control*.

Implementation The body control application has a port which is used to communicate with the light controller. There are two variants for the connector of this port: a low end and an adaptive curve variant. These two variants are exclusive.

A system constant `CarType` is used to switch between the two variants; a value of `1` means “low end” while a value of `2` means “adaptive curve”.

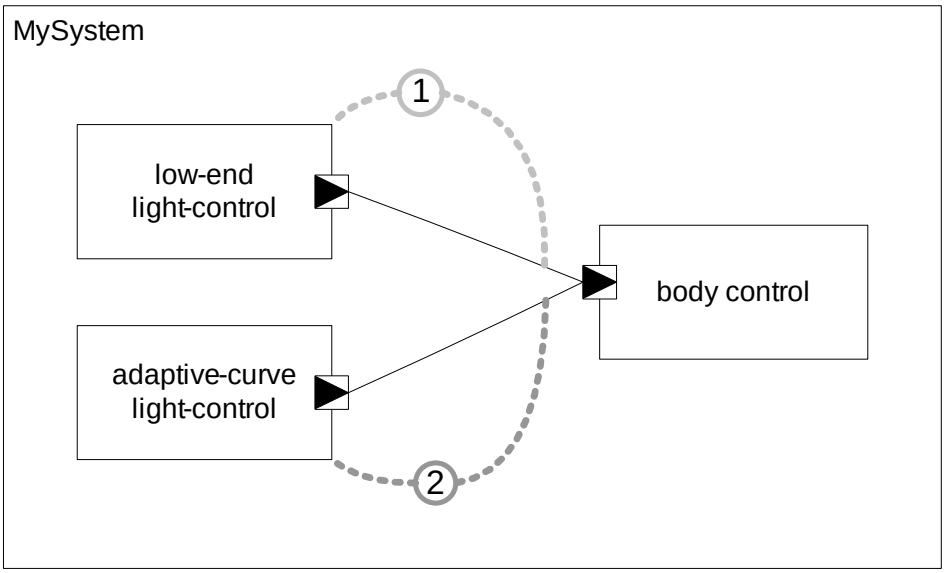

Figure 7.14: Example for *Aggregation Pattern*

Listing 7.2: Example for aggregation pattern in ARXML

```
<AR-PACKAGE>
  <SHORT-NAME>AggregationPattern</SHORT-NAME>
  <INTRODUCTION>
    <P>
      <L-1
        L="EN">Note that the example slightly varies from
        the example in the document: Body control has one port
        which is used for both variants. </L-1>
    </P>
  </INTRODUCTION>
  <REFERENCE-BASES>
    <REFERENCE-BASE>
      <SHORT-LABEL>default</SHORT-LABEL>
      <IS-DEFAULT>true</IS-DEFAULT>
      <IS-GLOBAL>false</IS-GLOBAL>
      <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
      <PACKAGE-REF
        DEST="AR-PACKAGE">/AggregationPattern</PACKAGE-REF>
    </REFERENCE-BASE>
  </REFERENCE-BASES>
  <ELEMENTS>
    <SW-SYSTEMCONST>
```

```
  <SHORT-NAME>CarType</SHORT-NAME>
</SW-SYSTEMCONST>
<COMPOSITION-SW-COMPONENT-TYPE>
  <SHORT-NAME>MySystem</SHORT-NAME>
  <COMPONENTS>
    <SW-COMPONENT-PROTOTYPE>
      <SHORT-NAME>LowEndLightControl</SHORT-NAME>
      <TYPE-TREF
        DEST="APPLICATION-SW-COMPONENT-TYPE">LowEndLightControl</
          TYPE-TREF>
      <VARIATION-POINT>
        <SHORT-LABEL>LowEnd</SHORT-LABEL>
        <SW-SYSCOND
          BINDING-TIME="SYSTEM-DESIGN-TIME">
          <SYSC-REF
          DEST="SW-SYSTEMCONST">CarType</SYSC-REF>
          == 1
        </SW-SYSCOND>
      </VARIATION-POINT>
    </SW-COMPONENT-PROTOTYPE>
    <SW-COMPONENT-PROTOTYPE>
      <SHORT-NAME>AdaptiveCurveLightControl</SHORT-NAME>
      <TYPE-TREF
        DEST="APPLICATION-SW-COMPONENT-TYPE">
          AdaptiveCurveLightControl</TYPE-TREF>
      <VARIATION-POINT>
        <SHORT-LABEL>HighEnd</SHORT-LABEL>
        <SW-SYSCOND
          BINDING-TIME="SYSTEM-DESIGN-TIME">
          <SYSC-REF
          DEST="SW-SYSTEMCONST">CarType</SYSC-REF>
          == 2
        </SW-SYSCOND>
      </VARIATION-POINT>
    </SW-COMPONENT-PROTOTYPE>
    <SW-COMPONENT-PROTOTYPE>
      <SHORT-NAME>BodyControl</SHORT-NAME>
      <TYPE-TREF
        DEST="APPLICATION-SW-COMPONENT-TYPE">BodyControl</TYPE-TREF
          >
    </SW-COMPONENT-PROTOTYPE>
  </COMPONENTS>
  <CONNECTORS>
    <ASSEMBLY-SW-CONNECTOR>
      <SHORT-NAME>LightControl</SHORT-NAME>
      <INTRODUCTION>
        <P>
          <L-1
            L="EN">Note that the shortname in both variants is
            the same.</L-1>
        </P>
      </INTRODUCTION>
      <VARIATION-POINT>
        <SHORT-LABEL>LowEnd</SHORT-LABEL>
        <SW-SYSCOND
          BINDING-TIME="SYSTEM-DESIGN-TIME">
```

```
            <SYSC-REF
            DEST="SW-SYSTEMCONST">CarType</SYSC-REF>
            == 1
          </SW-SYSCOND>
        </VARIATION-POINT>
        <PROVIDER-IREF>
          <CONTEXT-COMPONENT-REF
            DEST="SW-COMPONENT-PROTOTYPE">LowEndLightControl</CONTEXT
              -COMPONENT-REF>
          <TARGET-P-PORT-REF
            DEST="P-PORT-PROTOTYPE">ToLight</TARGET-P-PORT-REF>
        </PROVIDER-IREF>
        <REQUESTER-IREF>
          <CONTEXT-COMPONENT-REF
            DEST="SW-COMPONENT-PROTOTYPE">BodyControl</CONTEXT-
              COMPONENT-REF>
          <TARGET-R-PORT-REF
            DEST="R-PORT-PROTOTYPE">LightControlInput</TARGET-R-PORT-
              REF>
        </REQUESTER-IREF>
      </ASSEMBLY-SW-CONNECTOR>
      <ASSEMBLY-SW-CONNECTOR>
        <SHORT-NAME>LightControl</SHORT-NAME>
        <VARIATION-POINT>
          <SHORT-LABEL>HighEnd</SHORT-LABEL>
          <SW-SYSCOND
            BINDING-TIME="SYSTEM-DESIGN-TIME">
            <SYSC-REF
            DEST="SW-SYSTEMCONST">CarType</SYSC-REF>
            == 2
          </SW-SYSCOND>
        </VARIATION-POINT>
        <PROVIDER-IREF>
          <CONTEXT-COMPONENT-REF
            DEST="SW-COMPONENT-PROTOTYPE">AdaptiveCurveLightControl</
              CONTEXT-COMPONENT-REF>
          <TARGET-P-PORT-REF
            DEST="P-PORT-PROTOTYPE">ToLight</TARGET-P-PORT-REF>
        </PROVIDER-IREF>
        <REQUESTER-IREF>
          <CONTEXT-COMPONENT-REF
            DEST="SW-COMPONENT-PROTOTYPE">BodyControl</CONTEXT-
              COMPONENT-REF>
          <TARGET-R-PORT-REF
            DEST="R-PORT-PROTOTYPE">LightControlInput</TARGET-R-PORT-
              REF>
        </REQUESTER-IREF>
      </ASSEMBLY-SW-CONNECTOR>
    </CONNECTORS>
  </COMPOSITION-SW-COMPONENT-TYPE>
  <APPLICATION-SW-COMPONENT-TYPE>
    <SHORT-NAME>LowEndLightControl</SHORT-NAME>
    <PORTS>
      <P-PORT-PROTOTYPE>
        <SHORT-NAME>ToLight</SHORT-NAME>
      </P-PORT-PROTOTYPE>
```

```
      </PORTS>
    </APPLICATION-SW-COMPONENT-TYPE>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>AdaptiveCurveLightControl</SHORT-NAME>
      <PORTS>
        <P-PORT-PROTOTYPE>
          <SHORT-NAME>ToLight</SHORT-NAME>
        </P-PORT-PROTOTYPE>
      </PORTS>
    </APPLICATION-SW-COMPONENT-TYPE>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>BodyControl</SHORT-NAME>
      <PORTS>
        <R-PORT-PROTOTYPE>
          <SHORT-NAME>LightControlInput</SHORT-NAME>
        </R-PORT-PROTOTYPE>
      </PORTS>
    </APPLICATION-SW-COMPONENT-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

Listing 7.2 contains two pairs of VARIATION-POINT elements: one that switches between the alternative component prototypes, and one that switches between the alternative connectors.

The following excerpt from the above XML shows the code for the three component prototypes *LowEndLightControl* , *AdaptiveCurveLightControl* and *BodyControl*:

Listing 7.3: Variant Component Prototypes

```
<COMPONENTS>
  <SW-COMPONENT-PROTOTYPE>
    <SHORT-NAME>LowEndLightControl</SHORT-NAME>
    <TYPE-TREF
      DEST="APPLICATION-SW-COMPONENT-TYPE">LowEndLightControl</
          TYPE-TREF>
    <VARIATION-POINT>
      <SHORT-LABEL>LowEnd</SHORT-LABEL>
      <SW-SYSCOND
        BINDING-TIME="SYSTEM-DESIGN-TIME">
        <SYSC-REF
        DEST="SW-SYSTEMCONST">CarType</SYSC-REF>
        == 1
      </SW-SYSCOND>
    </VARIATION-POINT>
  </SW-COMPONENT-PROTOTYPE>
  <SW-COMPONENT-PROTOTYPE>
    <SHORT-NAME>AdaptiveCurveLightControl</SHORT-NAME>
    <TYPE-TREF
      DEST="APPLICATION-SW-COMPONENT-TYPE">
          AdaptiveCurveLightControl</TYPE-TREF>
    <VARIATION-POINT>
      <SHORT-LABEL>HighEnd</SHORT-LABEL>
      <SW-SYSCOND
        BINDING-TIME="SYSTEM-DESIGN-TIME">
        <SYSC-REF
```

```
            DEST="SW-SYSTEMCONST">CarType</SYSC-REF>
            == 2
          </SW-SYSCOND>
        </VARIATION-POINT>
      </SW-COMPONENT-PROTOTYPE>
      <SW-COMPONENT-PROTOTYPE>
        <SHORT-NAME>BodyControl</SHORT-NAME>
        <TYPE-TREF
          DEST="APPLICATION-SW-COMPONENT-TYPE">BodyControl</TYPE-TREF
            >
      </SW-COMPONENT-PROTOTYPE>
    </COMPONENTS>
```

Both *LowEndLightControl* and *AdaptiveCurveLightControl* contain a VARIATION-POINT element, which checks the value of the system constant named CarType.

There are similar variation points embedded in the description of the assembly connector prototypes. Note that these variation points have identical SHORT-NAMEs (*LightControl*), but use different SHORT-LABELs (*LowEnd* and *HighEnd*), as described in Section 7.6.2.

7.9.2 Example for *Association Pattern*

Description Software component type with variant diagnostics procedures.

Use Case The light controller from the previous example provides a standard and a high end method for diagnostics.

Implementation There are two diagnostics procedures: standard_light_diagnostics_proc and complex_diagnostics_proc. These procedures realize different algorithms, which require different API function calls. standard_light_diagnostics_proc is always called, complex_diagnostics_proc only in high-end vehicles.

The system constant *CarType* discriminates between the two types of vehicles: a value of 1 means standard car, a value of 2 high end car.

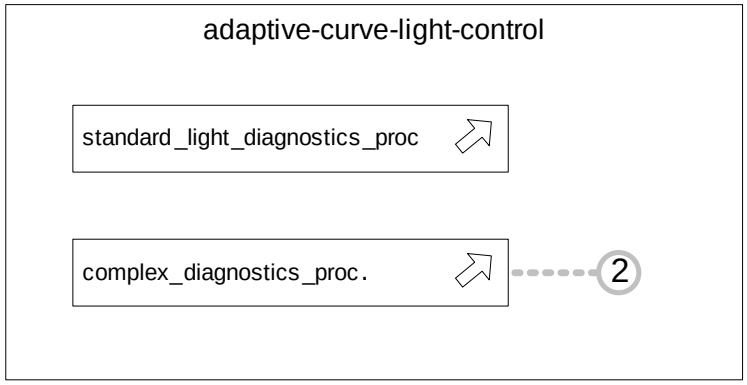

Figure 7.15: Example for *association pattern*

Listing 7.4: Example for association pattern in ARXML

```
<AR-PACKAGE>
  <SHORT-NAME>AssociationPattern</SHORT-NAME>
  <REFERENCE-BASES>
    <REFERENCE-BASE>
      <SHORT-LABEL>default</SHORT-LABEL>
      <IS-DEFAULT>true</IS-DEFAULT>
      <IS-GLOBAL>false</IS-GLOBAL>
      <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
      <PACKAGE-REF
        DEST="AR-PACKAGE">/AssociationPattern</PACKAGE-REF>
    </REFERENCE-BASE>
  </REFERENCE-BASES>
  <ELEMENTS>
    <BSW-MODULE-ENTRY>
      <SHORT-NAME>standard_light_diagnostics_proc</SHORT-NAME>
    </BSW-MODULE-ENTRY>
    <BSW-MODULE-ENTRY>
      <SHORT-NAME>complex_diagnostics_proc</SHORT-NAME>
    </BSW-MODULE-ENTRY>
    <BSW-MODULE-DESCRIPTION>
      <SHORT-NAME>MyDiagnosticManager</SHORT-NAME>
      <INTERNAL-BEHAVIORS>
        <BSW-INTERNAL-BEHAVIOR>
          <SHORT-NAME>Behavior</SHORT-NAME>
          <ENTITYS>
            <BSW-SCHEDULABLE-ENTITY>
              <SHORT-NAME>TheMainFunc</SHORT-NAME>
              <INTRODUCTION>
                <P>
                  <L-1
                    L="EN">complex_diagnostics_proc is called
                    in case of Highend. standard_light_diagnostics_proc
                    is always called.</L-1>
                </P>
              </INTRODUCTION>
              <CALLED-ENTRYS>
                <BSW-MODULE-ENTRY-REF-CONDITIONAL>
                  <BSW-MODULE-ENTRY-REF
                    DEST="BSW-MODULE-ENTRY">complex_diagnostics_proc</
                      BSW-MODULE-ENTRY-REF>
                  <VARIATION-POINT>
                    <SW-SYSCOND
                      BINDING-TIME="CODE-GENERATION-TIME">
                      <SYSC-REF
                      DEST="SW-SYSTEMCONST">/AggregationPattern/CarType
                          </SYSC-REF>
                      == 2
                    </SW-SYSCOND>
                  </VARIATION-POINT>
                </BSW-MODULE-ENTRY-REF-CONDITIONAL>
                <BSW-MODULE-ENTRY-REF-CONDITIONAL>
                  <BSW-MODULE-ENTRY-REF
```

```
                    DEST="BSW-MODULE-ENTRY">
                      standard_light_diagnostics_proc</BSW-MODULE-
                      ENTRY-REF>
                  </BSW-MODULE-ENTRY-REF-CONDITIONAL>
                </CALLED-ENTRYS>
              </BSW-SCHEDULABLE-ENTITY>
            </ENTITYS>
          </BSW-INTERNAL-BEHAVIOR>
        </INTERNAL-BEHAVIORS>
      </BSW-MODULE-DESCRIPTION>
    </ELEMENTS>
  </AR-PACKAGE>
```

The variation in Listing 7.4 is contained in the part where the `BswModuleEntity` calls the `BswModuleEntry`:

Listing 7.5: Example for association pattern in ARXML

```
<CALLED-ENTRYS>
  <BSW-MODULE-ENTRY-REF-CONDITIONAL>
    <BSW-MODULE-ENTRY-REF
      DEST="BSW-MODULE-ENTRY">complex_diagnostics_proc</
        BSW-MODULE-ENTRY-REF>
    <VARIATION-POINT>
      <SW-SYSCOND
        BINDING-TIME="CODE-GENERATION-TIME">
        <SYSC-REF
        DEST="SW-SYSTEMCONST">/AggregationPattern/CarType
          </SYSC-REF>
        == 2
      </SW-SYSCOND>
    </VARIATION-POINT>
  </BSW-MODULE-ENTRY-REF-CONDITIONAL>
  <BSW-MODULE-ENTRY-REF-CONDITIONAL>
    <BSW-MODULE-ENTRY-REF
      DEST="BSW-MODULE-ENTRY">
        standard_light_diagnostics_proc</BSW-MODULE-
        ENTRY-REF>
  </BSW-MODULE-ENTRY-REF-CONDITIONAL>
</CALLED-ENTRYS>
```

`BswModuleEntity` has a variant reference to `BswModuleEntry` (under the role name `calledEntry`). Hence, `<BSW-MODULE-ENTITY>` contains *two* elements named `<BSW-MODULE-ENTRY-REF-CONDITIONAL>`, each of which contains a reference to the called entry. The first one includes a `<VARIATION-POINT>`.

If the reference from `BswModuleEntity` to `BswModuleEntry` would not be variable, then there could be only a single element named `<BSW-MODULE-ENTRY-REF>`, and of course there would be no `<VARIATION-POINT>`. But otherwise, the code would look the same.

7.9.3 Example for *Attribute Value Pattern*

Description Adaptive algorithm for a variant array size

Use Case An engine control system needs to adapt to a wide range of engines. Especially, the number of cylinders may vary from engine to engine. The implementation needs to store data for each cylinder.

Implementation Data for the cylinders is stored in an array, who's size corresponds to the number of cylinders. Since we are in an embedded system, we cannot use dynamic arrays, but must use a static array, e.g. `CylinderData cylinder_measures[2*NO_OF_CYLINDERS]`. In the meta model, the size of the array (`2*NO_OF_CYLINDERS`) is an attribute of a class. This attribute must be variant.

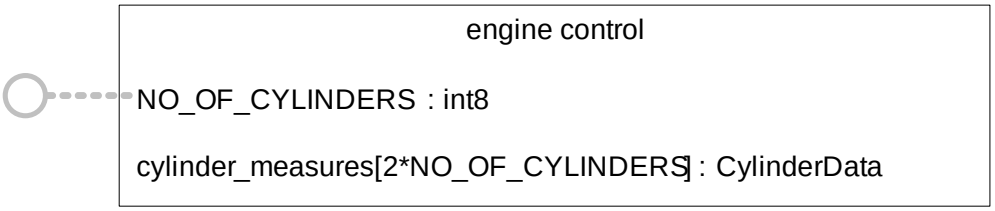

Figure 7.16: Example for *attribute value pattern*

Listing 7.6: Example for attribute value pattern in ARXML

```
<AR-PACKAGE>
  <SHORT-NAME>AttributeValuePattern</SHORT-NAME>
  <REFERENCE-BASES>
    <REFERENCE-BASE>
      <SHORT-LABEL>default</SHORT-LABEL>
      <IS-DEFAULT>true</IS-DEFAULT>
      <IS-GLOBAL>false</IS-GLOBAL>
      <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
      <PACKAGE-REF
        DEST="AR-PACKAGE">/AttributeValuePattern</PACKAGE-REF>
    </REFERENCE-BASE>
  </REFERENCE-BASES>
  <ELEMENTS>
    <SW-SYSTEMCONST>
      <SHORT-NAME>NO_OF_CYLINDERS</SHORT-NAME>
    </SW-SYSTEMCONST>
    <APPLICATION-RECORD-DATA-TYPE>
      <SHORT-NAME>CylinderMeasure</SHORT-NAME>
    </APPLICATION-RECORD-DATA-TYPE>
    <APPLICATION-ARRAY-DATA-TYPE>
      <SHORT-NAME>Cylinder_measures</SHORT-NAME>
      <ELEMENT>
        <SHORT-NAME>Cylinder_Measure</SHORT-NAME>
        <CATEGORY>VALUE</CATEGORY>
        <TYPE-TREF
          DEST="APPLICATION-DATA-TYPE">CylinderMeasure</TYPE-TREF>
        <MAX-NUMBER-OF-ELEMENTS
          BINDING-TIME="CODE-GENERATION-TIME">
          2 * <SYSC-REF
          DEST="SW-SYSTEMCONST">NO_OF_CYLINDERS</SYSC-REF>
```

```
          </MAX-NUMBER-OF-ELEMENTS>
        </ELEMENT>
      </APPLICATION-ARRAY-DATA-TYPE>
    </ELEMENTS>
  </AR-PACKAGE>
```

In this example, the variation lies in the size of the array `cylinder_measures`. Normally, this size would be a value; in this example, it is a reference to a system constant:

```
<MAX-NUMBER-OF-ELEMENTS BINDING-TIME="CODE-GENERATION-TIME">
 2 * <SYSC-REF DEST="SW-SYSTEMCONST">NO_OF_CYLINDERS</SYSC-REF>
</MAX-NUMBER-OF-ELEMENTS>
```

As one can see, there is no tag `VARIATION-POINT`. The variation point can be recognized by the attribute `BINDING-TIME`[22].

7.9.4 Example for *Property Set Pattern*

Description Varying properties of a `FlexrayCommunicationController`.

Use Case An ECU is connected to a FlexRay bus in two different configurations (car types), requiring different `FlexrayCommunicationController` settings.

Implementation Several property values of a `FlexrayCommunicationController` need to be consistently set or modified in one configuration step.

In our example, we have two variants for the attributes `microPerCycle`, `microtickDuration`, and `samplesPerMicrotick` of the `FlexrayCommunicationController`. Since these three attributes are mutually dependent in both configurations, and their values need to be set together.

The system constant `CarType` is used to discriminate between the two variants:

	CarType 1	CarType 2
`microPerCycle`	320000	80000
`microtickDuration`	$50E-9$	$12.5E-9$
`samplesPerMicrotick`	4	1

All other attributes are non-variant in our example. We show only the (alphabetically) first and last attributes of `FlexrayCommunicationController`, namely `acceptedStartupRange` and `wakeUpPattern` in the XML file, omitting the others for brevity.

This example is illustrated in Figure 7.17.

[22] Consequently, a formula without a `BINDING-TIME` attribute on the enclosing XML element is an error.

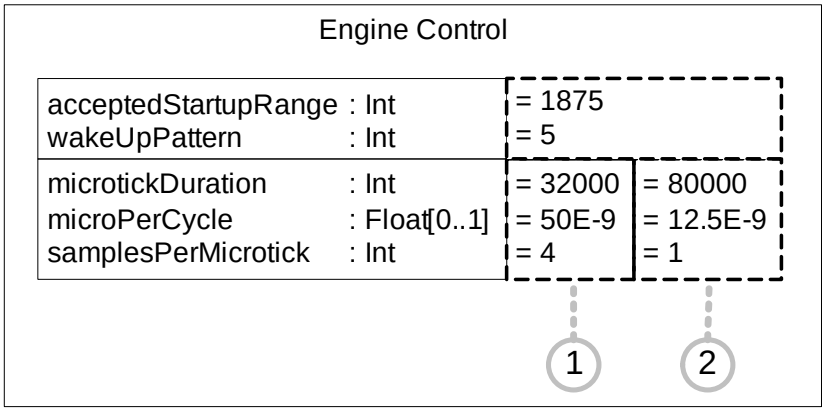

Figure 7.17: Example for *property set pattern*

Listing 7.7 shows the XML code for our example.

Listing 7.7: Example for property set pattern

```
<AR-PACKAGE>
  <SHORT-NAME>PropertySetPattern</SHORT-NAME>
  <ELEMENTS>
    <ECU-INSTANCE>
      <SHORT-NAME>iPhone</SHORT-NAME>
      <COMM-CONTROLLERS>
        <FLEXRAY-COMMUNICATION-CONTROLLER>
          <SHORT-NAME>iFlex</SHORT-NAME>
          <FLEXRAY-COMMUNICATION-CONTROLLER-VARIANTS>
            <FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
              <ACCEPTED-STARTUP-RANGE>1875</ACCEPTED-STARTUP-RANGE>
              <WAKE-UP-PATTERN>5</WAKE-UP-PATTERN>
            </FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
            <FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
              <MICRO-PER-CYCLE>320000</MICRO-PER-CYCLE>
              <MICROTICK-DURATION>50E-9</MICROTICK-DURATION>
              <SAMPLES-PER-MICROTICK>4</SAMPLES-PER-MICROTICK>
              <VARIATION-POINT>
                <SW-SYSCOND
                  BINDING-TIME="CODE-GENERATION-TIME">
                      <SYSC-REF
                  DEST="SW-SYSTEMCONST">/AggregationPattern/CarType</
                      SYSC-REF>
                        == 1
                    </SW-SYSCOND>
              </VARIATION-POINT>
            </FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
            <FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
              <MICRO-PER-CYCLE>80000</MICRO-PER-CYCLE>
              <MICROTICK-DURATION>12.5E-9</MICROTICK-DURATION>
              <SAMPLES-PER-MICROTICK>1</SAMPLES-PER-MICROTICK>
              <VARIATION-POINT>
                <SW-SYSCOND
                  BINDING-TIME="CODE-GENERATION-TIME">
                      <SYSC-REF
                  DEST="SW-SYSTEMCONST">/AggregationPattern/CarType</
                      SYSC-REF>
```

```
                    == 2
                  </SW-SYSCOND>
                </VARIATION-POINT>
              </FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
            </FLEXRAY-COMMUNICATION-CONTROLLER-VARIANTS>
          </FLEXRAY-COMMUNICATION-CONTROLLER>
        </COMM-CONTROLLERS>
      </ECU-INSTANCE>
    </ELEMENTS>
  </AR-PACKAGE>
```

At the heart of example 7.7 is the following construct, which defines three sets of variant attributes for the `FlexrayCommunicationController`.

```
<FLEXRAY-COMMUNICATION-CONTROLLER>
  <SHORT-NAME>iFlex</SHORT-NAME>
  <FLEXRAY-COMMUNICATION-CONTROLLER-VARIANTS>
    <FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
      ...
    </FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
    <FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
      ...
      <VARIATION-POINT>
        <SW-SYSCOND BINDING-TIME="CODE-GENERATION-TIME">
          <SYSC-REF DEST="SW-SYSTEMCONST">/AggregationPattern/CarType</SYSC
            -REF>
          == 1
        </SW-SYSCOND>
      </VARIATION-POINT>
    </FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
    <FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
      ...
      <VARIATION-POINT>
        <SW-SYSCOND BINDING-TIME="CODE-GENERATION-TIME">
          <SYSC-REF DEST="SW-SYSTEMCONST">/AggregationPattern/CarType</SYSC
            -REF>
          == 2
        </SW-SYSCOND>
      </VARIATION-POINT>
    </FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL>
  </FLEXRAY-COMMUNICATION-CONTROLLER-VARIANTS>
</FLEXRAY-COMMUNICATION-CONTROLLER>
```

As one can see, `FLEXRAY-COMMUNICATION-CONTROLLER` contains a wrapper element `FLEXRAY-COMMUNICATION-CONTROLLER-VARIANTS`, and a `FLEXRAY-COMMUNICATION-CONTROLLER-CONDITIONAL` for each variation:

- The first `CONDITIONAL` holds the non-variant attributes (`acceptedStartupRange` to `wakeUpPattern`). Note that this `CONDITIONAL` does not contain a variation point.
- The second `CONDITIONAL` holds the first set of variant attributes, namely those for *CarType* 1.

- The third CONDITIONAL holds the first set of variant attributes, namely those for *CarType* 2.

7.10 Definition of Binding Times

Table 7.30 presents an overview of the binding times that are supported by AUTOSAR Variant Handling.

Binding Time	Supported by AUTOSAR Variant Handling
BlueprintDerivationTime	some features of variation points are also used here
FunctionDesignTime	no
SystemDesignTime	yes
CodeGenerationTime	yes
PreCompileTime	yes
LinkTime	yes
PostBuild	yes
Runtime	no

Table 7.30: Binding Times

Variant Handling in AUTOSAR supports the following binding times: BlueprintDerivationTime (limited), SystemDesignTime, CodeGenerationTime, PreCompileTime, LinkTime, and PostBuild.

7.10.1 BlueprintDerivationTime

At BlueprintDerivationTime, a model is derived from Blueprints. For example, a function design tool provides the option to derive objects from a predefined set of blueprints. See [9] for more details. Strictly speaking, this is not variant handling, but some meta model features for variant handling also apply here (see Section 7.6.11).

BlueprintDerivationTime is out of the scope of this document, but mentioned here for completeness.

Output: model

7.10.2 FunctionDesignTime

At FunctionDesignTime, a software architecture independent model for (control) systems is developed. Typical tools used at this stage are *Matlab/Simulink*, or *ASCET-MD*.

If a function design tool supports variant handling according to AUTOSAR it has no other choice than using CodeGenerationTime or later as binding time in the generated AUTOSAR artifacts.

FunctionDesignTime is out of the scope of this document (as long as it does not affect calibration measurements), but mentioned here for completeness.

Output: model

7.10.3 SystemDesignTime

SystemDesignTime is characterized by the following tasks:

- Designing the VFB
- Software Component types (Interfaces)
- SWC Prototypes and the Connections between SWCprototypes
- Designing the Topology
- ECUs and interconnecting Networks
- Designing the Communication Matrix and Data Mapping

Input: Model from function design time, Requirements

Output: AUTOSAR model

7.10.4 CodeGenerationTime

At this step, code is generated. This may be done either by hand, or using a tool, or a mixture of both.

Handwritten code is typically based on a requirements document, whereas generated code is usually created from a model that was designed at FunctionDesignTime or SystemDesignTime.

Both the requirements and the model may contain variants, but code is only generated for those variants that have been selected, or which need to be resolved later.

Input: Model

Output: Program code

7.10.5 PreCompileTime

At PreCompileTime, a preprocessor (e.g., the C Preprocessor) is used to further customize the code and exclude parts of the code from the compilation process.

There are several reasons for such an exclusion: code is not required for the selected variant(s), code is incompatible with the selected variant(s), or code requires resources

that are not present in the selected variant(s). The code that is excluded at this stage code will not be available at later stages.

`PreCompileTime` is typically used for handwritten code (for which `SystemDesign-Time` and `CodeGenerationTime` obviously cannot not take effect) or when a system constant needs to be bound after code generation.

Input: Code

Output: Object code. Object code is only generated for the selected variant(s), or for variants.

7.10.6 LinkTime

The configuration at this stage determines which modules are included in the resulting object code (executable), and which ones are omitted based on the selected variants.

Input: Object code

Output: Object code (executable program?)

7.10.7 `PostBuild`

PostBuild is the binding time which is bound latest at startup of the ECU. In other words this is everything between creation of the executable program and startup of the ECU.

The startup of the ECU is the PostBuild binding since and obviously cannot be re12 solved in the model.

Input: Configuration data set

Output: –

7.10.8 Runtime

Everything after startup and initialization is `RunTime`. Variant Handling at `RunTime` is out of the scope of this document, but mentioned here for completeness.

8 Documentation Support

8.1 Introduction

The AUTOSAR meta-model and XML schema are based upon ASAM FSX[16]. They provide support for well-structured, content-focused and integrated documentation.

The documentation supported by AUTOSAR is "well structured" in the sense that the building blocks used to build the documentation delimit the beginning and the end of each construct, the constructs are context-neutral and can be cut and pasted wherever the construct is legal[1].

The author of an AUTOSAR documentation does not need to focus on formatting of the text. Instead, he/she can focus on the content, structuring this content in a hierarchy of chapters. He can use figures, lists, tables, and formula to express his though clearly, even making notes[2] to stress some points. However, he does not specify the appearance of these elements. He lets a tool format all of them according to a consistent style (e.g. corporate identity).

One advantages of such a well-structured and content-focused documentation is that it can be processed and transformed easily, thus supporting operation like merging different sources to produce a comprehensive documentation from multiple components' documentation. Different parts of the content can be annotated with their intended audience, so that it is possible to generate audience-specific documents from the same composite source.

The documentation can be integrated within an M1-level artifact (e.g., a SWC-T instance) in the following levels:

- A *single paragraph* (see Chapter 8.2.1) in the role `desc` provides a short description of an identifiable object to help a human being identify the object. It can be inserted in any `Identifiable` element (see section 4.4 for description of identifiable).
- A *documentation block* (see Chapter 8.2) is available in any `Identifiable` element as `introduction`. This type of documentation is typically used to capture a short introduction about the role of an element or respectively how it is built.
- An *standalone documentation* structured into multiple chapters is also offered in AUTOSAR. It supports references to particular model elements (see section 8.3 for details). This type of documentation is suited to describe the interactions among multiple elements and to provide a complete overview.

[1] The construct chapter, for example, can be nested into any other chapter becoming a sub-chapter or a sub-sub-chapter. The advantage becomes clear when one tries to restructure a LaTeX report moving chapters, sections, and subsections at different level in the document hierarchy.

[2] Examples of notes include caution, hints, exercises.

Figure 8.1 illustrates[3] the macroscopic text model of AUTOSAR, in particular the available entry points on which documentation is integrated in particular AUTOSAR metamodel elements. DocumentationContainer is not a real metamodel element. It is used in this diagram to illustrate any AUTOSAR metamodel element which may contain documentation.

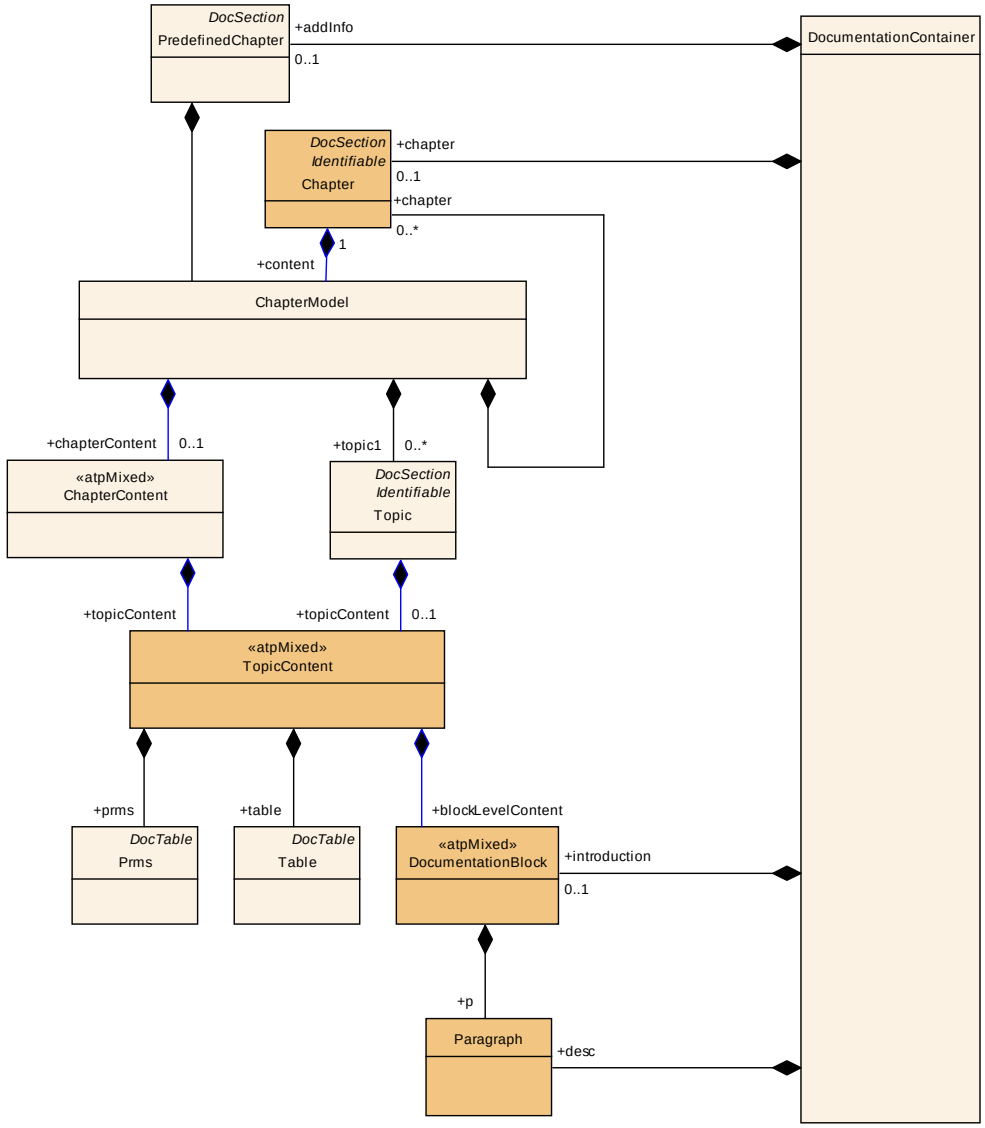

Figure 8.1: Macroscopic text model (simplified)

In figure 8.1, each of the three levels extends the mechanisms offered by the previous one. The documentation block used in the `introduction` contains paragraphs used in `desc`. The standalone documentation are build around chapters containing documentation block as well as generic tables `table` and parameter tables `prms`. There are two types of chapters: the `Chapter`, which have a title and the predefined chapter `PredefinedChapter`, which name is given by their context[4]. A chapter addresses

[3] note that this is a simplified illustration of the real model

[4] In the software component template, for example, a component has an optional set of predefined documentation aspects (e.g., swFeatureDesc, swTestDesc, swCalibrationNotes)

one or more topics. It may be organized into explicit topic constructs or directly into tables and documentation blocks. A `Topic` is a logical unit, which is given explicit boundaries and a label concisely capturing its content.

Currently chapters and topics have basically the same content. However, a chapter is typically formatted differently, may offer additional representation means and will probably be extended independently of the topic in the future. Therefore, `ChapterContent` contains the `TopicContent`

Please note that the various names (e.g., addInfo, prms, desc, introduction) are used to remain consistent with [16].

The remaining sections of this chapter describe the following aspects of AUTOSAR documentation:

- Section 8.2.1 describes the mechanisms used within a single paragraph.
- Section 8.2 describes the documentation features supported by the `introduction` and the classes used to implement them.
- Section 8.3 describes the exhaustive documentation features used to produce a standalone documentation.
- Section 8.6 describes the multi-language support offered by AUTOSAR documentation.

8.2 Documentation Block

The `DocumentationBlock` (see figure 8.2) is limited such that it can be represented in a table cell and contains the following basic documentation elements:

- **paragraph**: The construct `p` identifies the text organized as a paragraph.
- **List**: A list is a sequence of items. In AUTOSAR, the concept of list implemented using different keywords. This concept is explained below (i.e., after this list).
- **Figure**: The `figure` is the construct used to insert a diagram in a document. It is composed of a caption (short-name and long-name) and the diagram itself. Depending on the tool used, different diagram format are supported (example of graphical format include svg (scalable vector graphic), jpeg, and gif).
- **Formula**: The construct `formula` is used to specify a mathematical expression. Different kinds of specification of the formula are possible (for example the TeX math format).
- **Verbatim**: `verbatim` represents a block in which whitespace (in particular blanks and line feeds) are kept “as they are”. This enables basic formatting to be carried out, which can even be displayed on simple devices. Verbatim is often used to represent source code or xml in books.

- **note**: The construct `note` is used to express an annotation. Examples of annotation types include hint, caution, tip, instruction, and exercise.
- **trace**: This is used to manage tracing between items in documents. See [9] for more details.
- **structuredReq**: this is used for requirements documents. See [9] for more details.

Class	≪**atpMixed**≫ **DocumentationBlock**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements			
Note	This class represents a documentation block. It is made of basic text structure elements which can be displayed in a table cell.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
defList	DefList	0..1	aggr	This represents a definition list in the documentation block. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.sequenceOffset=40
figure	MlFigure	0..1	aggr	This represents a figure in the documentation block. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.sequenceOffset=70
formula	MlFormula	0..1	aggr	This is a formula in the definition block. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.sequenceOffset=60
labeledList	LabeledList	0..1	aggr	This represents a labeled list. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.sequenceOffset=50
list	List	0..1	aggr	This represents numbered or unnumbered list. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.sequenceOffset=30
note	Note	0..1	aggr	This represents a note in the text flow. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.sequenceOffset=80
p	MultiLanguageParagraph	0..1	aggr	This is one particular paragraph. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.sequenceOffset=10

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
structured Req	StructuredReq	0..1	aggr	This aggregation supports structured requirements embedded in a documentation block. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.sequenceOffset=100
trace	TraceableText	0..1	aggr	This represents traceable text in the documentation block. This allows to specify requirements/constraints in any documentation block. The kind of the trace is specified in the category. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.sequenceOffset=90
verbatim	MultiLanguageVerbatim	0..1	aggr	This represents one particular verbatim text. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.sequenceOffset=20

Table 8.1: DocumentationBlock

Figure 8.2 illustrates the constructs contained in a documentation block and the constructs that contains a documentation block. In this figure the containment of each construct is a `atpVariation` with a latest binding time (see 7.10).

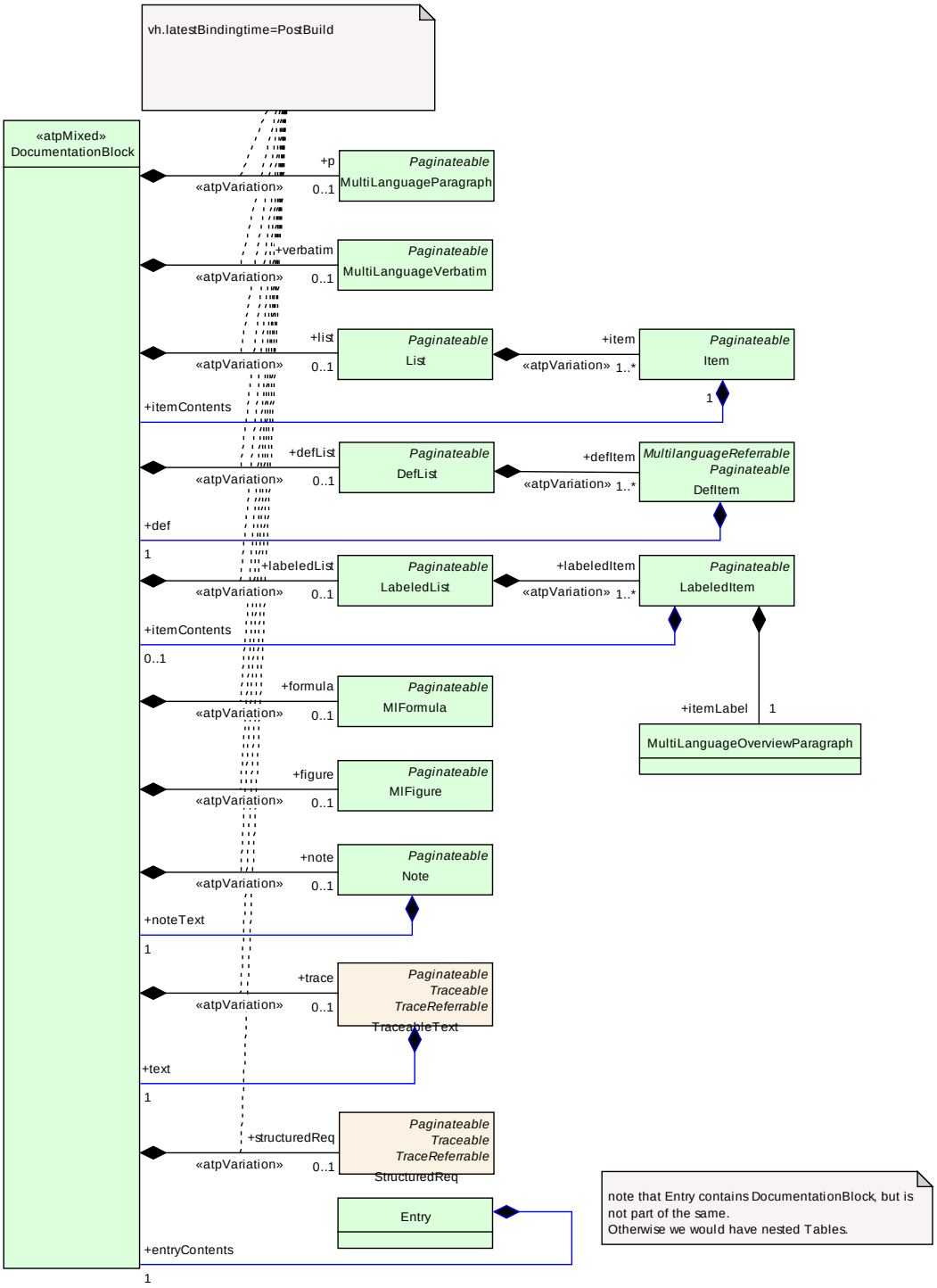

Figure 8.2: documentation block overview

8.2.1 Paragraph

There are two different types of paragraphs in AUTOSAR. They differ only in the constructs, which they may contain: the overview paragraph (`LOverviewParagraph`)

used in `desc` and the full blown paragraph (`LParagraph`) used in documentation block.

Within both types of paragraphs, it is possible to use the constructs described in the tables below to create footnotes, subscripts, superscripts, emphasis, and line break, as well inserting index entries, reference targets and references. The `LParagraph` offers the additional capability to make references to external files, documents, and standards. This capability is not offered in `LOverviewParagraph`.

Class	≪`atpMixedString`≫ **MixedContentForOverviewParagraph (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineTextModel			
Note	This is the text model of a restricted paragraph item within a documentation. Such restricted paragraphs are used mainly for overview items, e.g. desc.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
br	Br	1	aggr	This element is the same as function here as in a HTML document i.e. it forces a line break.
e	EmphasisText	1	aggr	This is emphasis text. **Tags:** xml.sequenceOffset=60
ft	FootnoteText	1	aggr	This is text printed as footnote. **Tags:** xml.sequenceOffset=70
ie	IndexEntry	1	aggr	This is an index entry. **Tags:** xml.sequenceOffset=100
sub	Superscript	1	attr	This is superscript text. **Tags:** xml.sequenceOffset=90
sup	Superscript	1	attr	This is subscript text. **Tags:** xml.sequenceOffset=80
tt	Tt	1	aggr	This is a technical term. **Tags:** xml.sequenceOffset=30
xref	Xref	1	aggr	This is a cross reference. **Tags:** xml.sequenceOffset=40
xrefTarget	XrefTarget	1	aggr	This element specifies a reference target which can be scattered throughout the text. **Tags:** xml.sequenceOffset=50

Table 8.2: MixedContentForOverviewParagraph

Class	**MultiLanguageParagraph**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::MultilanguageData			
Note	This is the content model of a multilingual paragraph in a documentation.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax must be defined by the applied help system respectively help system generator. **Tags:** xml.attribute=true
l1	LParagraph	1..*	aggr	This is the paragraph content in one partiucular language. **Tags:** xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false

Table 8.3: MultiLanguageParagraph

Class	≪`atpMixedString`≫ **MixedContentForParagraph (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineTextModel			
Note	This mainly represents the text model of a full blown paragraph within a documentation.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
br	Br	1	aggr	This element is the same as function here as in a HTML document i.e. it forces a line break. **Tags:** xml.sequenceOffset=40
e	EmphasisText	1	aggr	This is emphasized text. **Tags:** xml.sequenceOffset=70
ft	FootnoteText	1	aggr	This is a footnote within a parapgraph. **Tags:** xml.sequenceOffset=80
ie	IndexEntry	1	aggr	This is an index entry. **Tags:** xml.sequenceOffset=110
std	Std	1	aggr	This is a refeernce to a standard. **Tags:** xml.sequenceOffset=120
sub	Superscript	1	attr	This is subscript text. **Tags:** xml.sequenceOffset=100
sup	Superscript	1	attr	This is superscript text. **Tags:** xml.sequenceOffset=90

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
tt	Tt	1	aggr	This is a technical term. **Tags:** xml.sequenceOffset=30
xdoc	Xdoc	1	aggr	This is a reference to a printable external document. **Tags:** xml.sequenceOffset=130
xfile	Xfile	1	aggr	This represents a reference to an external file which usually cannot be printed. **Tags:** xml.sequenceOffset=140
xref	Xref	1	aggr	This is a cross reference. **Tags:** xml.sequenceOffset=50
xrefTarget	XrefTarget	1	aggr	This element specifies a reference target which can be scattered throughout the text. **Tags:** xml.sequenceOffset=60

Table 8.4: MixedContentForParagraph

8.2.2 Verbatim

Verbatim (represented by `MultiLanguageVerbatim` respectively `MixedContentForVerbatim`) marks the text as "preformatted" – all the spaces and carriage returns are rendered exactly as they are typed be the user respectively appear in the model. Such content should also be rendered using a mono spaced font.

Even if the content needs to be rendered verbatim, the model still allows inline elements. It is expected the the number of characters are not changed when those elements are rendered.

Class	**MultiLanguageVerbatim**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::MultilanguageData			
Note	This class represents multilingual Verbatim. Verbatim means, that white-space is maintained. When Verbatim is rendered in PDF or Online media, white-space is obeyed. Blanks are rendered as well as newline characters.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
allowBreak	NameToken	0..1	attr	This indicates if the verbatim text might be split on multiple pages. Default is "1". **Tags:** xml.attribute=true
float	FloatEnum	0..1	attr	Indicate whether it is allowed to break the element. The following values are allowed: **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax must be defined by the applied help system respectively help system generator. **Tags:** xml.attribute=true
l5	LVerbatim	1..*	aggr	This the text in one particular language. **Tags:** xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false
pgwide	PgwideEnum	0..1	attr	Used to indicate wether the figure should take the complete page width (value = "pgwide") or not (value = "noPgwide"). **Tags:** xml.attribute=true

Table 8.5: MultiLanguageVerbatim

Class	≪`atpMixedString`≫ **MixedContentForVerbatim (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineTextModel			
Note	This is the text model for preformatted (verbatim) text. It mainly consists of attributes which do not change the length on rendering. This class represents multilingual verbatim. Verbatim, sometimes called preformatted text, means that white-space is maintained. When verbatim is rendered in PDF or Online media, it is rendered using a monospaced font while white-space is obeyed. Blanks are rendered as well as newline characters. Even if there are inline elements, the length of the data must not be influenced by formatting.			
Base	ARObject,WhitespaceControlled			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
br	Br	1	aggr	This element is the same as function here as in a HTML document i.e. it forces a line break. **Tags:** xml.sequenceOffset=50
e	EmphasisText	1	aggr	This is emphsized text. Note that in verbatim, the attribute font should not be considered since verbatim is always rendered as monospace font. **Tags:** xml.sequenceOffset=30
tt	Tt	1	aggr	This represents a technical term in verbatim. Note that it's the responibility of the user not to take a tt that would add additional character to the text (such as SgmlElement).

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
xref	Xref	1	aggr	This is a crossreference within a verbatim text. The attributes may disturb the arrangement of the text. It is subject to the author to keep this under control. **Tags:** xml.sequenceOffset=40

Table 8.6: MixedContentForVerbatim

Class	**WhitespaceControlled (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::LanguageDataModel			
Note	This meta-class represents the ability to control the white-space handling e.g. in xml serialization. This is implemented by adding the attribute "space".			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
xmlSpace	XmlSpaceEnum	1	attr	This attribute 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. **Tags:** xml.attribute=true; xml.attributeRef=true; xml.enforceMinMultiplicity=true; xml.name=space; xml.nsPrefix=xml

Table 8.7: WhitespaceControlled

8.2.3 Lists in Documentation

In documentation it is often appropriate to present facts in form of lists. AUTOSAR supports the following kinds of lists:

- The **plain list** (`List`) is composed of one or more list items (`item`). There are two types of lists, numbered and unnumbered[5] This is controlled by `type` in `List`.

  An `Item` contains a `DocumentationBlock`. In most cases, it is a simple paragraph, but it often one or more paragraphs followed by a sub-list.

- The **labeled list** (`LabeledList`) is a list, where every item has a label (`LabeledItem`) and a content, which is a `DocumentationBlock`.

  The policy how to render the labeled list is denoted in `itemLabelPos` in `IndentSample`

- The **definition list** (`DefList`) is used to introduce terms and their definition. The term is captured in a `DefItem` and the definition expressed in a `DocumentationBlock`.

[5] An unnumbered list is also called bullet list.

Note that the `DefList` maintains the specific semantics of defintions, even if it might be rendered in the same way as `LabeledList`. For example the `defItem`s in a `DefList` are `Referrable`.

[constr_2520] Nesting of lists shall be limited ⌈ The nesting of lists shall be limited to a reasonable depth such that it can safely be rendered on A4 pages. A reasonable approach is not to nest more than three levels. ⌋

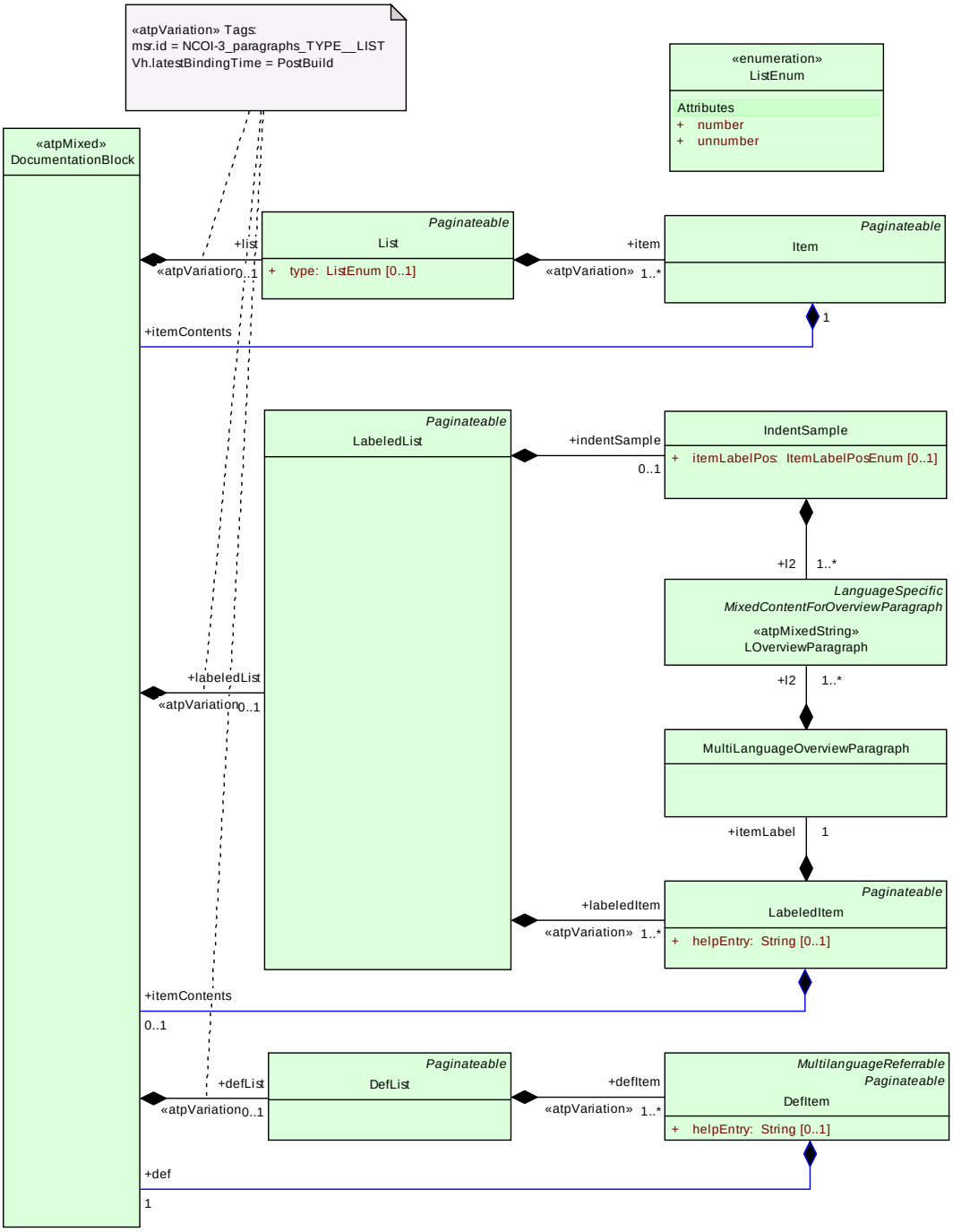

Figure 8.3: List Element Overview

8.2.3.1 Class tables for `List`

Class	**Item**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::ListElements			
Note	This meta-class represents one particular item in a list.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
itemContents	DocumentationBlock	1	aggr	this represents the actual content of the item. It is composed of a DocumentationBlock. This way it is possible to use simple paragraphs to nested lists, formula, figures or notes. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.typeElement=false; xml.typeWrapperElement=false

Table 8.8: Item

Enumeration	**ListEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::ListElements
Note	This meta-class represents the notation of the various types of lists.
Literal	***Description***
number	This indicates that the list is an numerated list.
unnumber	This indicates that it is an enumeration (bulleted list)

Table 8.9: ListEnum

8.2.3.2 Class tables for `LabeledList`

Class	**LabeledList**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::ListElements			
Note	This meta-class represents a labeled list, in which items have a label and a content. The policy how to render such items is specified in the labeled list.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
indentSample	IndentSample	0..1	aggr	This is a sample item. This sample is used by a rendering system to measure out the width of indentation. Since this depends on the particular fontsize etc. the indentation cannot be specified e.g. in mm. **Tags:** xml.sequenceOffset=20

Attribute	Datatype	Mul.	Kind	Note
labeledItem	LabeledItem	1..*	aggr	This represents one particular item in the labeled list. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=30; xml.type Element=false; xml.typeWrapperElement=false

Table 8.10: LabeledList

Class	LabeledItem			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::ListElements			
Note	this represents an item of a labeled list.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax must be defined by the applied help system respectively help system generator. **Tags:** xml.attribute=true
itemContents	DocumentationBlock	0..1	aggr	This represents the actual content of the item. It is composed of a DocumentationBlock. This way it is possible to use simple paragraphs to nested lists, formula, figures or notes. **Tags:** xml.roleElement=false; xml.roleWrapper Element=false; xml.sequenceOffset=30; xml.type Element=false; xml.typeWrapperElement=false
itemLabel	MultiLanguage OverviewParagraph	1	aggr	This is the label of the item. **Tags:** xml.sequenceOffset=20

Table 8.11: LabeledItem

Class	IndentSample			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::ListElements			
Note	This represents the ability to specify indentation of a labeled list by providing a sample content. This content can be measured by the rendering system in order to determine the width of indentation.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
itemLabelPos	ItemLabelPosEnum	0..1	attr	The position of the label in case the label is too long. The default is "NO-NEWLINE" **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
l2	LOverviewParagraph	1..*	aggr	This represents the indent sample in one particular language. **Tags:** xml.roleElement=true; xml.roleWrapperElement=false; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false

Table 8.12: IndentSample

Enumeration	**ItemLabelPosEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::ListElements
Note	This enumerator specifies, how the label of a labeled list shall be rendered.
Literal	*Description*
newline	The label is renders in a new line.
newlineIfNecessary	The label is rendered in a new line if it is longer than the indentation.
noNewline	The label is rendered in one line with the item even if it is longer than the indentation.

Table 8.13: ItemLabelPosEnum

8.2.3.3 Class tables for `DefList`

Class	**DefList**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::ListElements			
Note	This meta-class represents the ability to express a list of definitions. Note that a definition list might be rendered similar to a labeled list but has a particular semantics to denote definitions.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
defItem	DefItem	1..*	aggr	This is one entry in the definition list. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.roleElement=true; xml.roleWrapperElement=false; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false

Table 8.14: DefList

Class	**DefItem**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::ListElements			
Note	This represents an entry in a definition list. The defined item is specified using shortName and longName.			
Base	ARObject,DocumentViewSelectable,MultilanguageReferrable,Paginateable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
def	Documentation Block	1	aggr	This represents the definition part of the DefItem. **Tags:** xml.sequenceOffset=20
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax must be defined by the applied help system respectively help system generator. **Tags:** xml.attribute=true

Table 8.15: DefItem

8.2.4 Figures in Documentation

AUTOSAR supports to include figures in documentation by `MlFigure`. This is composed of a caption (`figureCaption`) and a graphic `lGraphic` with language attribute. The caption gives a title to the figure and also makes the figure referrable).

The graph contains the actual diagram in a standardized diagram format as specified in `GraphicNotationEnum`.

The figure contains also a `Map` specifying regions of an image or object and assigning a specific action to each region (e.g., retrieve a document, run a program, etc.) When the region is activated by the user, the action is executed.

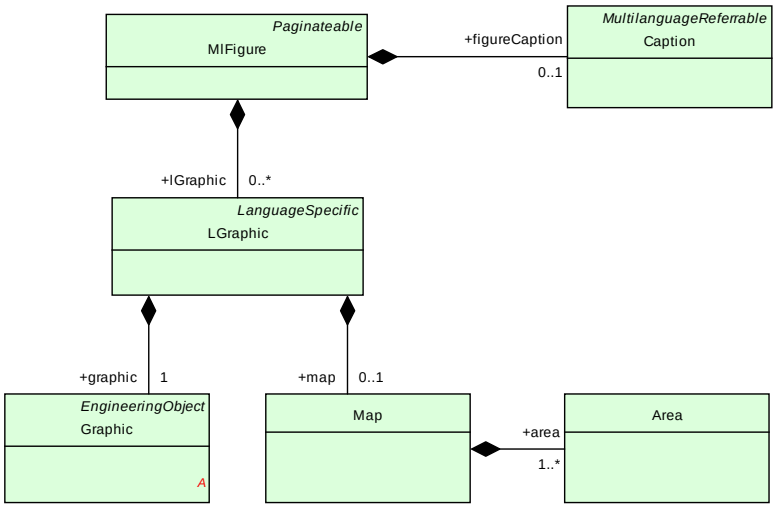

Figure 8.4: Figure Overview

Class	**Area**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Figure			
Note	This element specifies a region in an image map. Image maps enable authors to specify regions in an object (e.g. a graphic) and to assign a specific activity to each region (e.g. load a document, launch a program etc.). For more details refer to the specification of HTML.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
accesskey	String	0..1	attr	This attribute assigns an access key to an element. An access key is an individual character (e.g. "B") within the document character range. If an access key with an element assigned to it is pressed, the element comes into focus. The activity performed when an element comes into focus, is dependent on the element itself **Tags:** xml.attribute=true
alt	String	0..1	attr	This attribute specifies the text to be inserted as an alternative to illustrations, shapes or applets, where these cannot be displayed by user agents. **Tags:** xml.attribute=true
class	String	0..1	attr	Blank separated list of classes **Tags:** xml.attribute=true
coords	String	0..1	attr	This attribute specifies the position and shape on the screen. The number of values and their order depend on the geometrical figure defined. **Tags:** xml.attribute=true
href	String	0..1	attr	This attribute specifies the memory location of a web resource. It is therefore able to specify a link between the current element and the target element. **Tags:** xml.attribute=true
nohref	AreaEnumNohref	0..1	attr	If this attribute is set, the Area has no associated link. **Tags:** xml.attribute=true
onblur	String	0..1	attr	The ONBLUR-Event occurs, when focus is switched away from an element. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
onclick	String	0..1	attr	The ONCLICK-Event occurs, if the current element is clicked-on. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
ondblclick	String	0..1	attr	The ONCLICK-Event occurs, if the current element is "double" clicked-on. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
onfocus	String	0..1	attr	The ONFOCUS-Event occurs, if an element comes into focus (e.g., through navigation using the tab button). A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
onkeydown	String	0..1	attr	The ONKEYDOWN-Event occurs, if a button on the current element is pressed down. A script can be stored in this attribute to be performed in the event. **Tags:** xml.attribute=true
onkeypress	String	0..1	attr	The ONKEYPRESS-Event occurs, if a button on the current element is pressed down and released. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
onkeyup	String	0..1	attr	The ONKEYUP-Event occurs, if a button on the current element is released. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
onmousedown	String	0..1	attr	The ONMOUSEDOWN-Event occurs, if the mouse button used for clicking is held down on the current element. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
onmousemove	String	0..1	attr	The ONMOUSEMOVE-Event occurs, if the mouse pointer is moved on the current element (i.e. it is located on the current element). A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
onmouseout	String	0..1	attr	The ONMOUSEOUT-Event occurs, if the mouse pointer is moved from the current element. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
onmouseover	String	0..1	attr	The ONMOUSEOVER-Event occurs, if the mouse pointer is moved to the current element from another location outside it. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
onmouseup	String	0..1	attr	The ONMOUSEUP-Event occurs if the mouse button used for clicking is released on the current element. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
shape	AreaEnumShape	0..1	attr	The shape of the area. Note that in HTML this is defaulted to RECT. **Tags:** xml.attribute=true
style	String	0..1	attr	Information on the associated style **Tags:** xml.attribute=true
tabindex	String	0..1	attr	This attribute specifies the position of the current element in tabbing-order for the corresponding document. The value must lie between 0 and 32767. The Tabbing Order defines the sequence in which elements are focused on, when the user navigates using the keyboard. **Tags:** xml.attribute=true
title	String	0..1	attr	Title information of the Area element **Tags:** xml.attribute=true

Table 8.16: Area

Enumeration	**AreaEnumNohref**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Figure
Note	This enumerator specifies the fact that the area has no reference.
Literal	***Description***
nohref	This indicates that the area has no active link.

Table 8.17: AreaEnumNohref

Enumeration	**AreaEnumShape**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Figure
Note	This enumerator specifies the shape of the area.
Literal	***Description***
circle	The shape is a circle.
default	This specifies the fact that the area covers the rest of the figure.
poly	The area is specified as polygon.
rect	The shape is specified as rectangle.

Table 8.18: AreaEnumShape

Class	**Graphic**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Figure			
Note	This class represents an artifact containing the image to be inserted in the document			
Base	ARObject,EngineeringObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
editHeight	String	0..1	attr	Specifies the height of the graphic when it is displayed in an editor. **Tags:** xml.attribute=true
editWidth	String	0..1	attr	Specifies the width of the graphic when it is displayed in an editor. **Tags:** xml.attribute=true
editfit	GraphicFitEnum	0..1	attr	Specifies how the graphic shall be displayed in an editor. If the attribute is missing, **Tags:** xml.attribute=true
editscale	String	0..1	attr	Set the proportional scale when displayed in an editor. **Tags:** xml.attribute=true
filename	String	0..1	attr	Name of the file that should be displayed. This attribute is supported in ASAM FSX and kept in AUTOSAR in order to support cut and paste. **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
fit	GraphicFitEnum	0..1	attr	It determines the way in which the graphic should be inserted. Enter the attribute value "AS-IS" , to insert a graphic in its original dimensions. The graphic is adapted, if it is too big for the space for which it was intended. Default is "AS-IS" **Tags:** xml.attribute=true
generator	NameToken	0..1	attr	This attribute specifies the generator which is used to generate the image. Use case is that when editing a documentation, a figure (to be delivered by the modeling tool) is inserted by the authoring tool as reference (this is the role of graphic). But the real figure maybe injected during document processing. To be able to recognize this situation, this attribute can be applied. **Tags:** xml.attribute=true
height	String	0..1	attr	Define the displayed height of the figure. **Tags:** xml.attribute=true
htmlFit	GraphicFitEnum	0..1	attr	How to fit the graphic in an online media. Default is AS-IS. **Tags:** xml.attribute=true
htmlHeight	String	0..1	attr	Specifies the height of the graphic when it is displayed online. **Tags:** xml.attribute=true
htmlScale	String	0..1	attr	Set the proportional scale when displayed online. **Tags:** xml.attribute=true
htmlWidth	String	0..1	attr	Specifies the width of the graphic when it is displayed online. **Tags:** xml.attribute=true
notation	GraphicNotation Enum	0..1	attr	This attribute captures the format used to represent the graphic. **Tags:** xml.attribute=true; xml.id=GRAPHIC_TYPE __GRAPHIC_TYPE__NOTATION
scale	String	0..1	attr	In this element the dimensions of the graphic can be altered proportionally. **Tags:** xml.attribute=true
width	String	0..1	attr	Define the displayed width of the figure. **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Table 8.19: Graphic

Enumeration	**GraphicFitEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Figure
Note	This enumerator specifies the policy how to place and scale the figure on the page.
Literal	***Description***
AsIs	This indicates that the image shall be incorporated as is without scaling, rotation etc.
FitToPage	Fit to the page
FitToText	fit to the text containing the graphic.
LimitToPage	This indicates that the width of the graphic shall be limited to the **page width**. The image shall not be scaled down but cropped.
LimitToText	This indicates that the width of the graphic shall be limited to the width of the current **text flow**. The image shall not be scaled down but cropped.
Rotate180	Rotate 180 degree
Rotate180 LimitToText	Rotate 180 degree
Rotate90Ccw FitToText	Rotate by 90 degree counter clock wise and then fit to text
Rotate90Ccw LimitToText	Rotate by 90 degree counter clock wise and then fit to text
Rotate90Cw	Rotate 90 degree clockwise
Rotate90Cw FitToText	Rotate by 90 degree and then fit to text
Rotate90Cw LimitToText	Rotate by 90 degree and then fit to text
Rotate90ccw	Rotate 90 degree counter clockwise

Table 8.20: GraphicFitEnum

Enumeration	**GraphicNotationEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Figure
Note	This enumerator specifies the various notations (finally file types) used to represent the figure.
Literal	***Description***
eps	Encapsulated Postscript
gif	Graphics Interchange Format
jpg	"Joint Photographic Experts Group" format
svg	scalable vector graphic

Table 8.21: GraphicNotationEnum

Class	**Map**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Figure			
Note	Image maps enable authors to specify regions of an image or object and assign a specific action to each region (e.g., retrieve a document, run a program, etc.) When the region is activated by the user, the action is executed. The class follows the html approach and is intended to support interactive documents.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
area	Area	1..*	aggr	This element specifies a region in an image map. Image maps enable authors to specify regions in an object (e.g. a graphic) and to assign a specific activity to each region (e.g. load a document, launch a program etc.). **Tags:** xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false
class	String	0..1	attr	This attribute assigns a class name or set of class names to an element. Any number of elements may be assigned the same class name or set of class names. Multiple class names must be separated by white space characters. Class names are typically used to apply CSS formatting rules to an element. **Tags:** xml.attribute=true
name	NameToken	0..1	attr	This attribute assigns a name to the image map in the MAP element. This name can be used to be referenced in an HTML image through the attribute USEMAP. Although this is not actually necessary in the MSR model, it was inserted in order to support the MAPs which were created for HTML. **Tags:** xml.attribute=true
onclick	String	0..1	attr	The ONCLICK-Event occurs, if the current element is clicked on. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
ondblclick	String	0..1	attr	The ONDBLCLICK-Event occurs, if the current Event is "double" clicked-on. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
onkeydown	String	0..1	attr	The ONKEYDOWN-Event occurs, if a button on the current element is pressed down. A script can be stored in this attribute to be performed in the event. **Tags:** xml.attribute=true
onkeypress	String	0..1	attr	The ONKEYPRESS-Event occurs, if a button on the current element is pressed down and released. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
onkeyup	String	0..1	attr	The ONKEYUP-Event occurs, if a button on the current element is released. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
onmousedown	String	0..1	attr	The ONMOUSEDOWN-Event occurs, if the mouse button used for clicking is held down on the current element. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
onmousemove	String	0..1	attr	The ONMOUSEMOVE-Event occurs, if the mouse pointer is moved on the current element (i.e. it is located on the current element). A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
onmouseout	String	0..1	attr	The ONMOUSEOUT-Event occurs, if the mouse pointer is moved from the current element. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
onmouseover	String	0..1	attr	The ONMOUSEOVER-Event occurs, if the mouse pointer is moved to the current element from another location outside it. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
onmouseup	String	0..1	attr	The ONMOUSEUP-Event occurs if the mouse button used for clicking is released on the current element. A script can be stored in this attribute to be performed in the Event. **Tags:** xml.attribute=true
style	String	0..1	attr	This attribute specifies formatting style information for the current element. The content of this attribute is called inline CSS. The style attribute is deprecated (considered outdated), because it fuses together content and formatting. **Tags:** atp.Status=obsolete xml.attribute=true
title	String	0..1	attr	This attribute offers advisory information. Some Web browsers will display this information as tooltips. Authoring tools may make this information available to users as additional information about the element. **Tags:** xml.attribute=true

Table 8.22: Map

Class	**MIFigure**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Figure			
Note	This metaclass represents the ability to embed a figure.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
figureCaption	Caption	0..1	aggr	This element specifies the title of an illustration.
frame	FrameEnum	0..1	attr	Used to defined the frame line around a figure. It can assume the following values: • TOP - Border at the top of the figure • BOTTOM - Border at the bottom of the figure • TOPBOT - Borders at the top and bottom of the figure • ALL - Borders all around the figure • SIDES - Borders at the sides of the figure • NONE - No borders around the figure **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax must be defined by the applied help system respectively help system generator. **Tags:** xml.attribute=true
lGraphic	LGraphic	*	aggr	Container of the graphic (or diagram) and optional map of the figure in a given language. **Tags:** xml.roleWrapperElement=false; xml.sequenceOffset=30
pgwide	PgwideEnum	0..1	attr	Used to indicate wether the figure should take the complete page width (value = "pgwide") or not (value = "noPgwide"). **Tags:** xml.attribute=true
verbatim	MultiLanguageVerbatim	0..1	aggr	<verbatim> is a paragraph in which white-space (in particular blanks and line feeds) is obeyed. This enables basic preformatting to be carried out, which can even be displayed on simple devices. Behavior is the same as PRE in HTML . **Tags:** xml.sequenceOffset=50

Table 8.23: MlFigure

Class	**LGraphic**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Figure			
Note	This meta-class represents the figure in one particular language.			
Base	ARObject,LanguageSpecific			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
graphic	Graphic	1	aggr	Reference to the actual graphic represented in the figure. **Tags:** xml.sequenceOffset=20
map	Map	0..1	aggr	Image maps enable authors to specify regions of an image or object and assign a specific action to each region. **Tags:** xml.sequenceOffset=30

Table 8.24: LGraphic

8.2.5 Formula in Documentation

AUTOSAR supports to use formula to document mathematical subjects. `MlFormula` supports an optional caption (`formulaCaption`) to give a title to the formula and also makes the formula referable.

The formula itself takes one the following forms:

- a graphic (`LGraphic`), which contains the formula represented as a graphic.
- a math captured in TEX math mode (`texMath`)
- a generic math
- a verbatim (`verbatim`) when the formula is written as simple text, where the spaces are preserved

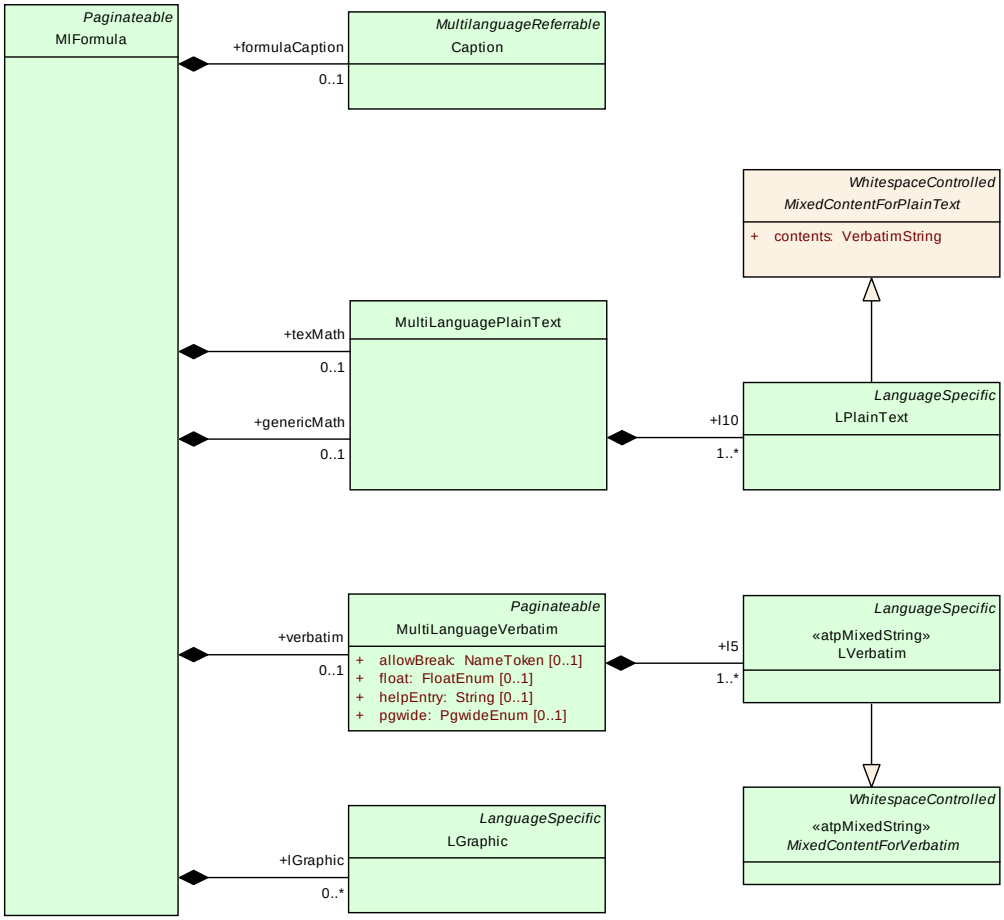

Figure 8.5: Formula Overview

Class	**MIFormula**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Formula			
Note	This meta-class represents the ability to express a formula in a documentation. The formula can be expressed by various means. If more than one representation is available, they need to be consistent. The rendering system can use the representation which is most appropriate.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	Datatype	Mul.	Kind	Note
formulaCaption	Caption	0..1	aggr	This element specifies the identification or heading of a formula. **Tags:** xml.sequenceOffset=20
genericMath	MultiLanguagePlainText	0..1	aggr	this rpresents the semantic and mathematical descriptions which are processed by a math-processor. **Tags:** xml.sequenceOffset=80
lGraphic	LGraphic	*	aggr	This represents a formula as an embedded figure. **Tags:** xml.roleWrapperElement=false; xml.sequenceOffset=30
texMath	MultiLanguagePlainText	0..1	aggr	this is the TeX representation of TeX formula. A TeX formula can be processed by a TeX or a LaTeX processor. **Tags:** xml.sequenceOffset=60
verbatim	MultiLanguageVerbatim	0..1	aggr	this represents a formula using only text and white-space. It can be used to denote the formula in a kind of pseudo code or whatever appears approprate. **Tags:** xml.sequenceOffset=50

Table 8.25: MIFormula

8.2.6 Notes in Documentation

The meta-class `Note` can be used to place notes with side heads and icons in a document. It is used for example to highlight instructions, exercises, cautions etc. The note itself contains a documentation block. It is composed of an optional label and one or more paragraphs.

[constr_2522] Notes should not be nested ⌈ Note even if it is possible to nest notes it is not recommended to do so, since it might lead to problems with the rendering of the note icon. ⌋

Class	Note			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Note			
Note	This represents a note in a documentation, which may be used to highlight specific issues such as hints or caution notes. N.B., Documentation notes can be nested recursively, even if this is not really intended. In case of nested notes e.g. the note icon of inner notes might be omitted while rendering the note.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
label	MultilanguageLongName	0..1	aggr	This label can be used to superseed the default label specified by the noteType attribute. It is in particular useful for noteType="other". **Tags:** xml.sequenceOffset=20
noteText	DocumentationBlock	1	aggr	This is the text content of the note. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=30; xml.typeElement=false; xml.typeWrapperElement=false
noteType	NoteTypeEnum	0..1	attr	Type of the Note. Default is "HINT" **Tags:** xml.attribute=true

Table 8.26: Note

Enumeration	**NoteTypeEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Note
Note	This enumerator specifies the type of the note. It can be used to render a note label or even a note icon.
Literal	***Description***
caution	This indicates that the note is an alert which shall be considered carefully.
example	This indicates that the note represents an example, e.g. a code example etc.
exercise	This indicates that the note represents an exercise for the reader.
hint	This indicates that the note represents a hint which helps the user for better understanding.
instruction	This indicates that the note represents an instruction, e.g. a step by step procedure.
other	This indicates that the note is something else. The particular type of the note shall then be specified in the label of the note.
tip	This indicates that the note represents which is good to know. It is similar to a hint, but focuses more to good practice than to better understanding.

Table 8.27: NoteTypeEnum

8.2.7 Support for Traceability in Documentation

AUTOSAR documentation support includes the ability to perform tracing between text elements. This tracing is primarily intended to be applied as bottom up tracing such as tracing from a specification statement to particular requirements which are fulfilled by the specified item.

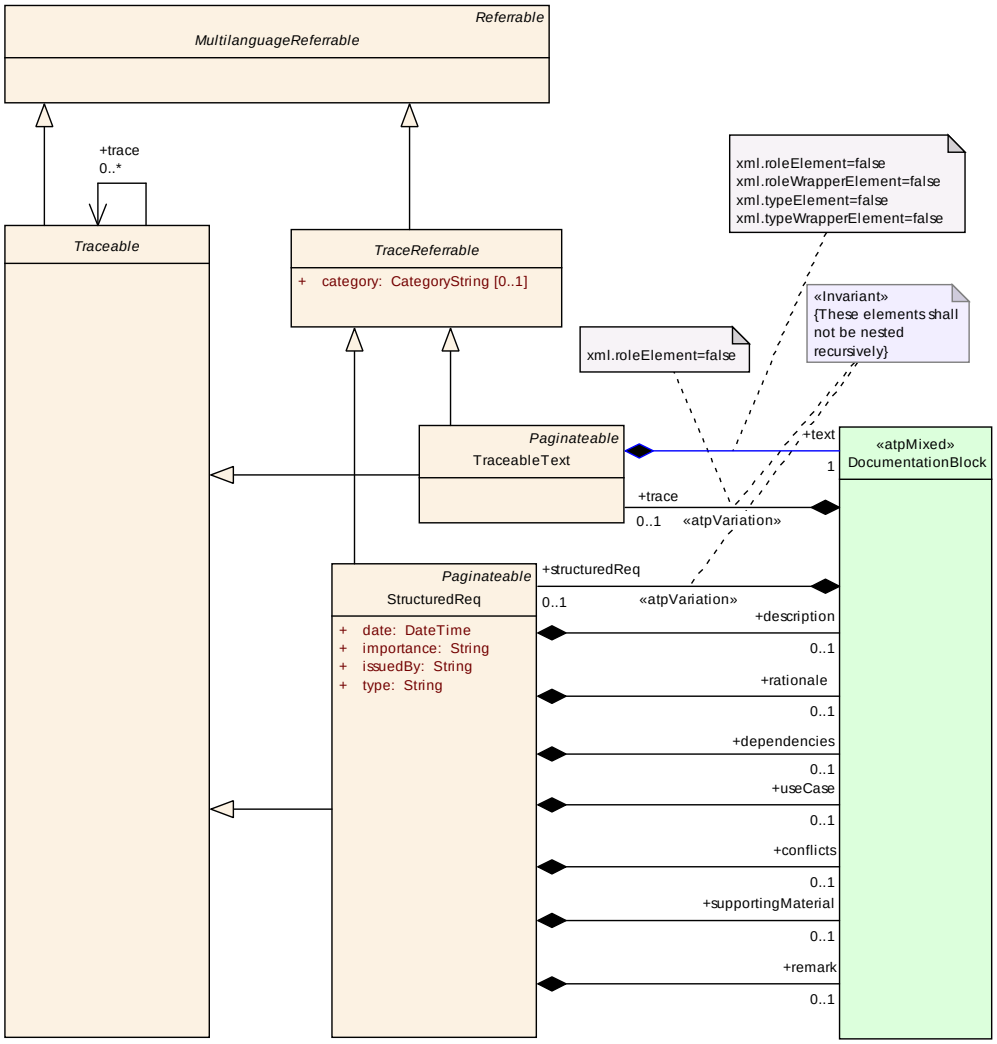

Figure 8.6: Support for Traceability

Class	**Traceable (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::RequirementsTracing			
Note	This meta class represents the ability to be subject to tracing within an AUTOSAR model. Note that it is expected that its subclasses inherit either from MultilanguageReferrable or from Identifiable. Nevertheless it also inherits from MultilanguageReferrable in order to provide a common reference target for all Traceables.			
Base	ARObject,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
trace	Traceable	*	ref	This assocation represents the ability to trace to upstream requirements / constraints. This supports for example the bottom up tracing ProjectObjectives <- MainRequirements <- Features <- RequirementSpecs <- BSW/AI **Tags:** xml.sequenceOffset=20

Attribute	Datatype	Mul.	Kind	Note

Table 8.28: Traceable

Class	TraceableText			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::RequirementsTracing			
Note	This meta-class represents the ability to denote a traceable text item such as requirements etc. The following approach appliles: • **shortName** represents the tag for tracing • **longName** represents the head line • **category** represents the kind of the tagged text			
Base	ARObject,DocumentViewSelectable,MultiLanguage Referrable,Paginateable,Referrable,TraceReferrable,Traceable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
text	Documentation Block	1	aggr	This represents the text to which the tag applies. **Tags:** xml.roleElement=false; xml.roleWrapper Element=false; xml.sequenceOffset=30; xml.type Element=false; xml.typeWrapperElement=false

Table 8.29: TraceableText

Class	StructuredReq			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::RequirementsTracing			
Note	This represents a structured requirement. This is intended for a case where specific requirements for features are collected. Note that this can be rendered as a labeled list.			
Base	ARObject,DocumentViewSelectable,MultiLanguage Referrable,Paginateable,Referrable,TraceReferrable,Traceable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
conflicts	Documentation Block	0..1	aggr	This represents an informal specification of conflicts. **Tags:** xml.sequenceOffset=40
date	DateTime	1	attr	**Tags:** xml.sequenceOffset=5
dependencies	Documentation Block	0..1	aggr	This represents an informal specifiaction of dependencies. Note that upstream tracing should be formalized in the property trace provided by the superclass Traceable. **Tags:** xml.sequenceOffset=30

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
description	Documentation Block	0..1	aggr	Ths represents the general description of the requirement. **Tags:** xml.sequenceOffset=10
importance	String	1	attr	This allows to represent the importance of the requirement. **Tags:** xml.sequenceOffset=8
issuedBy	String	1	attr	**Tags:** xml.sequenceOffset=6
rationale	Documentation Block	0..1	aggr	This represents the rationale of the requirement. **Tags:** xml.sequenceOffset=20
remark	Documentation Block	0..1	aggr	This represents an informal remark. Note that this is not modeled as annotation, since these remark is still essential part of the requirement. **Tags:** xml.sequenceOffset=60
supporting Material	Documentation Block	0..1	aggr	This represents an informal specifiaction of the supporting material. **Tags:** xml.sequenceOffset=50
type	String	1	attr	This attribute allows to denote the type of requirement to denote for example is it an "enhancement", "new feature" etc. **Tags:** xml.sequenceOffset=7
useCase	Documentation Block	0..1	aggr	This describes the relevant use cases. Note that formal references to use cases should be done in the trace relation. **Tags:** xml.sequenceOffset=35

Table 8.30: StructuredReq

8.2.8 Mixed Content and Inline Text Model Element

Depending on the context, AUTOSAR supports various inline elements. Inline elements represents specific markup of text within e.g. a paragraph. Example for this is subscript/superscript etc.

Figure 8.7 indicates the availability of inline elements in the various content models.

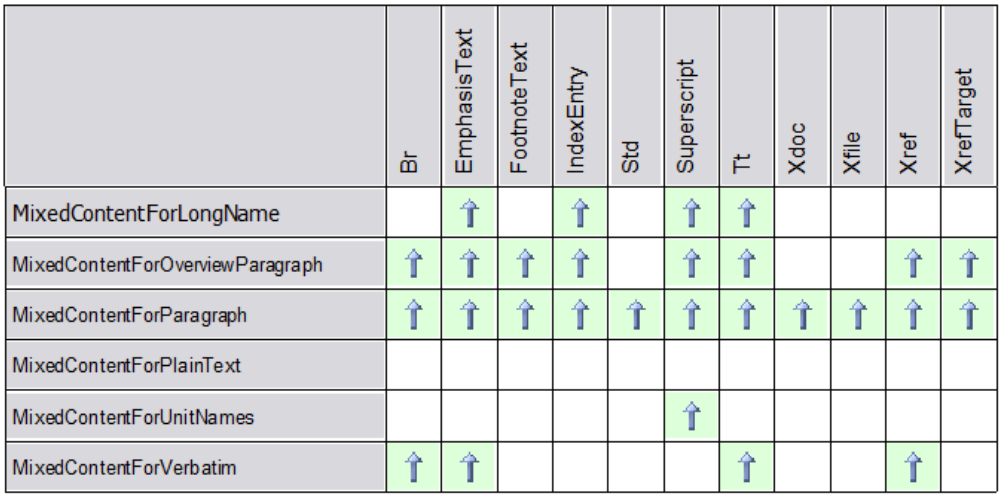

Figure 8.7: Inline text model

Class	**Br**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineTextElements			
Note	This element is the same as function here as in a HTML document i.e. it forces a line break.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 8.31: Br

Class	≪`atpMixedString`≫ **EmphasisText**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineTextElements			
Note	This is an emphasized text. As a compromise it contains some rendering oriented attributes such as color and font.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
color	String	0..1	attr	This allows to recommend a color of the emphasis. It is specified either as color name or as hexadecimal number. The syntax is the one of CSS, even if it is modeled as a string. **Tags:** xml.attribute=true
font	EEnumFont	0..1	attr	This specifies the font style in which the emphasized text shall be rendered. **Tags:** xml.attribute=true
sub	Superscript	1	attr	this is subscript text
sup	Superscript	1	attr	This is superscript text

Attribute	Datatype	Mul.	Kind	Note
type	EEnum	0..1	attr	Indicates how the text may be emphasized. Note that this is only a proposal which can be overridden or ignored by particular formatting engines. Default is BOLD. **Tags:** xml.attribute=true

Table 8.32: EmphasisText

Primitive	ExtIdClassEnum
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineAttributeEnums
Note	This is in fact an enumerator. The possible values are all legal XML names of identifiable objects even those of other XML files. If the schemas of all questionable files are generated from a common meta-model, this is something like an IdentifiableSubtypesEnum. Maybe a future version of the Schema generator can generate such an enum. As of now it is specified as string. **Tags:** xml.xsd.customType=EXT-ID-CLASS-ENUM; xml.xsd.type=string

Table 8.33: ExtIdClassEnum

Class	≪`atpMixedString`≫ FootnoteText			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineTextElements			
Note	This meta-class represents the ability to express text rendered as footnote.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
e	EmphasisText	1	aggr	This is an emphasized text. **Tags:** xml.sequenceOffset=40
sub	Superscript	1	attr	this is subscript text
sup	Superscript	1	attr	this is superscript text
tt	Tt	1	aggr	This is a technical term. **Tags:** xml.sequenceOffset=30

Table 8.34: FootnoteText

Class	≪`atpMixedString`≫ IndexEntry			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineTextElements			
Note	This class represents an index entry.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
sub	Superscript	1	attr	This is subscript text. **Tags:** xml.sequenceOffset=40
sup	Superscript	1	attr	This is superscript text. **Tags:** xml.sequenceOffset=30

Table 8.35: IndexEntry

Class	**Std**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses:: Documentation::TextModel::InlineTextElements			
Note	This represents a reference to external standards.			
Base	ARObject,Referrable,SingleLanguageReferrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
date	DateTime	0..1	attr	This element specifies the release date of the external standard if applicable. **Tags:** xml.sequenceOffset=50
position	String	0..1	attr	This represents the reference to the relevant positions of a standard. Kept as a string. **Tags:** xml.sequenceOffset=70
state	String	0..1	attr	This represents version and state of a standard. Kept as a string. **Tags:** xml.sequenceOffset=40
subtitle	String	0..1	attr	This represents the subtitle of the standard. **Tags:** xml.sequenceOffset=30
url	Url	0..1	aggr	This represents the URL of the standard. **Tags:** xml.sequenceOffset=60

Table 8.36: Std

Primitive	**Superscript**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses:: Documentation::TextModel::InlineTextElements
Note	This is text which is rendered superscript or subscript depending on the role. **Tags:** xml.xsd.customType=SUPSCRIPT; xml.xsd.type=string

Table 8.37: Superscript

Class	**Tt**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineTextElements			
Note	This meta-class represents the ability to express specific technical terms. The kind of term is denoted in the attribute "type".			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
term	String	1	attr	This is the term itself. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.typeElement=false; xml.typeWrapperElement=false
type	NameToken	1	attr	This attribute specifies the type of the technical term. Values are such as "VARIABLE" "CALPRM". It is no longer an enum in order to support process specific extensions. **Tags:** xml.attribute=true

Table 8.38: Tt

Class	**Xdoc**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineTextElements			
Note	This meta-class represents the ability to refer to an external document which can be rendered as printed matter.			
Base	ARObject,Referrable,SingleLanguageReferrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
date	DateTime	0..1	attr	This element specifies the release date of the external document if applicable. **Tags:** xml.sequenceOffset=50
number	String	0..1	attr	This represents document number of an external document that is referenced. Kept as a string. **Tags:** xml.sequenceOffset=30
position	String	0..1	attr	This represents the reference to the relevant positions of a standard. Kept as a string. **Tags:** xml.sequenceOffset=80
publisher	String	0..1	attr	This represents the publisher of an external document that is being referenced. Kept as a string. **Tags:** xml.sequenceOffset=60
state	String	0..1	attr	This represents version and state of the external document. Kept as a string. **Tags:** xml.sequenceOffset=40
url	Url	0..1	aggr	This specifies the URL of the external document. **Tags:** xml.sequenceOffset=70

Attribute	Datatype	Mul.	Kind	Note

Table 8.39: Xdoc

Class	Xfile			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineTextElements			
Note	This represents to reference an external file within a documentation.			
Base	ARObject,Referrable,SingleLanguageReferrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
tool	String	0..1	attr	This element describes the tool which was used to generate the corresponding Xfile . Kept as a string since no specific syntax can be provided to denote a tool. **Tags:** xml.sequenceOffset=50
toolVersion	String	0..1	attr	This element describes the tool version which was used to generate the corresponding xfile. Kept as a string, since no specific syntax can be specified. **Tags:** xml.sequenceOffset=60
url	Url	0..1	aggr	This represents the URL of the external file. **Tags:** xml.sequenceOffset=30

Table 8.40: Xfile

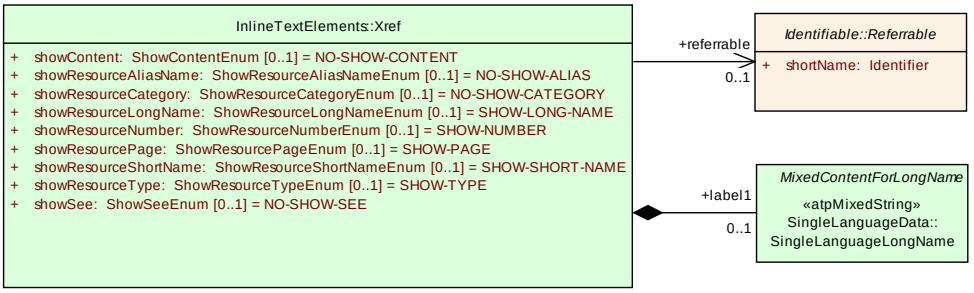

Figure 8.8: Xref overview

Class	Xref			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineTextElements			
Note	This represents a cross-reference within documentation.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
label1	SingleLanguage LongName	0..1	aggr	This allows to specify a replacement text which shall be rendered if showContent is selected.
referrable	Referrable	0..1	ref	This establishes the reference in Autosar style

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
showContent	ShowContentEnum	0..1	attr	Indicates if the content of the xref element shall be rendered. The default is "NO-SHOW-CONTENT". **Tags:** xml.attribute=true
showResourceAliasName	ShowResourceAliasNameEnum	0..1	attr	This indicates if the alias names of the referenced objects shall be rendered. This means this is some kind of backward searching: look whether there is an alias for the referenced object, if yes, print it. If there is more than one AliasNameSet, Xref might render all of those. If no alilas is found and showResourceShortName is set to NoShowShortName, then the shortName of the reference target shall be displayed. By this showResourceAliasName is similar to showResourceShortName but shows the aliasName instead of the shortName. Default is NO-SHOW-ALIAS-NAME. **Tags:** xml.attribute=true
showResourceCategory	ShowResourceCategoryEnum	0..1	attr	Indicates if the category of the referenced resource shall be rendered. Default is "NO-SHOW-CATEGORY". **Tags:** xml.attribute=true
showResourceLongName	ShowResourceLongNameEnum	0..1	attr	Indicates if the longName of the referenced resource shall be rendered. Default is "SHOW-LONG-NAME". **Tags:** xml.attribute=true
showResourceNumber	ShowResourceNumberEnum	0..1	attr	Indicates if the Number of the referenced resource shall be shown. Default is "SHOW–NUMBER" **Tags:** xml.attribute=true
showResourcePage	ShowResourcePageEnum	0..1	attr	Indicates if the page number of the referenced resource shall be shown. Default is "SHOW-PAGE" **Tags:** xml.attribute=true
showResourceShortName	ShowResourceShortNameEnum	0..1	attr	Indicates if the shortJName of the referenced resource shall be shown. Default is "SHOW-SHORT-NAME" **Tags:** xml.attribute=true
showResourceType	ShowResourceTypeEnum	0..1	attr	Indicates if the type of the referenced Resource shall be shown. Default is "SHOW-TYPE" **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
showSee	ShowSeeEnum	0..1	attr	Indicates if the word "see " shall be shown before the reference. Default is "NO-SHOW-SEE". Note that this is there for compatibility reasons only. **Tags:** xml.attribute=true

Table 8.41: Xref

Class	**XrefTarget**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineTextElements			
Note	This element specifies a reference target which can be scattered throughout the text.			
Base	ARObject,Referrable,SingleLanguageReferrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 8.42: XrefTarget

Enumeration	**EEnumFont**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineAttributeEnums
Note	This specifies the possible kind of fonts to be used for emphasis.
Literal	*Description*
default	The emphasis uses the default font.
mono	The emphasis uses a monospaced font.

Table 8.43: EEnumFont

Enumeration	**EEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineAttributeEnums
Note	This specifies the possible kinds of emphasis as proposal how to render it on paper or screen. Note that it would have been better to use plain, weak (italic), strong (bold), veryStrong (bolditalic) ... But users complained about this.
Literal	*Description*
bold	The emphasis is preferably represented in boldface font.
bolditalic	The emphasis is preferably represented in boldface plus italic font.
italic	The emphasis is preferably represented in italic font.
plain	The emphasis has no specific rendering. It is used if e.g. semantic information is applied to the emphasis text.

Table 8.44: EEnum

Enumeration	**ShowContentEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineAttributeEnums

Note	This specifies if the content of the xref element shall be rendered.
Literal	***Description***
noShow Content	The content of the Xref.label is **not** rendered at the place of the reference.
showContent	The content of the element is rendered at the place of the reference.

Table 8.45: ShowContentEnum

Enumeration	**ShowResourceAliasNameEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the alias names of the reference target shall be rendered with the xref.
Literal	***Description***
noShowAlias Name	This indicates that alias names of the referenced object shall **not** be rendered at the place of the reference.
showAlias Name	This indicates that the alias names of the referenced object shall be rendered at the place of the reference.

Table 8.46: ShowResourceAliasNameEnum

Enumeration	**ShowResourceCategoryEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the category of the reference target shall be rendered with the xref.
Literal	***Description***
noShow Category	The category of the target is **not** rendered at the place of the reference.
showCate-gory	The category of the target is rendered at the place of the reference.

Table 8.47: ShowResourceCategoryEnum

Enumeration	**ShowResourceLongNameEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the long name of the reference target shall be rendered with the xref.
Literal	***Description***
noShowLong Name	The long name of the target is **not** rendered at the place of the reference.
showLong Name	The long name of the target is rendered at the place of the reference.

Table 8.48: ShowResourceLongNameEnum

Enumeration	**ShowResourceNumberEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the number (e.g. chapter number) of the reference target shall be rendered with the xref.
Literal	***Description***
noShow Number	The number of the target is **not** rendered at the place of the reference.
showNumber	The number of the target is rendered at the place of the reference.

Table 8.49: ShowResourceNumberEnum

Enumeration	**ShowResourcePageEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the page numer of the reference target shall be rendered with the xref.
Literal	***Description***
noShowPage	The page number of the target is **not** rendered at the place of the reference.
showPage	The page number of the target is rendered at the place of the reference.

Table 8.50: ShowResourcePageEnum

Enumeration	**ShowResourceShortNameEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the short name of the reference target shall be rendered with the xref.
Literal	***Description***
noShowShort Name	The short name of the target is **not** rendered at the place of the reference.
showShort Name	The short name of the target is rendered at the place of the reference.

Table 8.51: ShowResourceShortNameEnum

Enumeration	**ShowResourceTypeEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the type (e.g. derived from the class) of the reference target shall be rendered with the xref.
Literal	***Description***
noShowType	The type of the target is **not** rendered at the place of the reference.
showType	The type of the target is rendered at the place of the reference.

Table 8.52: ShowResourceTypeEnum

Enumeration	**ShowSeeEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::InlineAttributeEnums
Note	This enumerator specifies if the word "see" shall be rendered before the xref.
Literal	***Description***
noShowSee	The word "see" is **not** rendered before the reference.
showSee	The word "see"is rendered before the reference.

Table 8.53: ShowSeeEnum

8.3 Standalone Documentation

The standalone documentation provides means to capture documentation independently of the structure of an AUTOSAR system. In order to to achieve this, it extends the introduction by adding chapters, topics, visual tables, and generic parameter sets (`prms`). One could say, it wraps the `DocumentationBlock` in chapters, topics, tables.

In addition to this, it allows to refer to AUTOSAR-Objects, which are the context of the documentation.

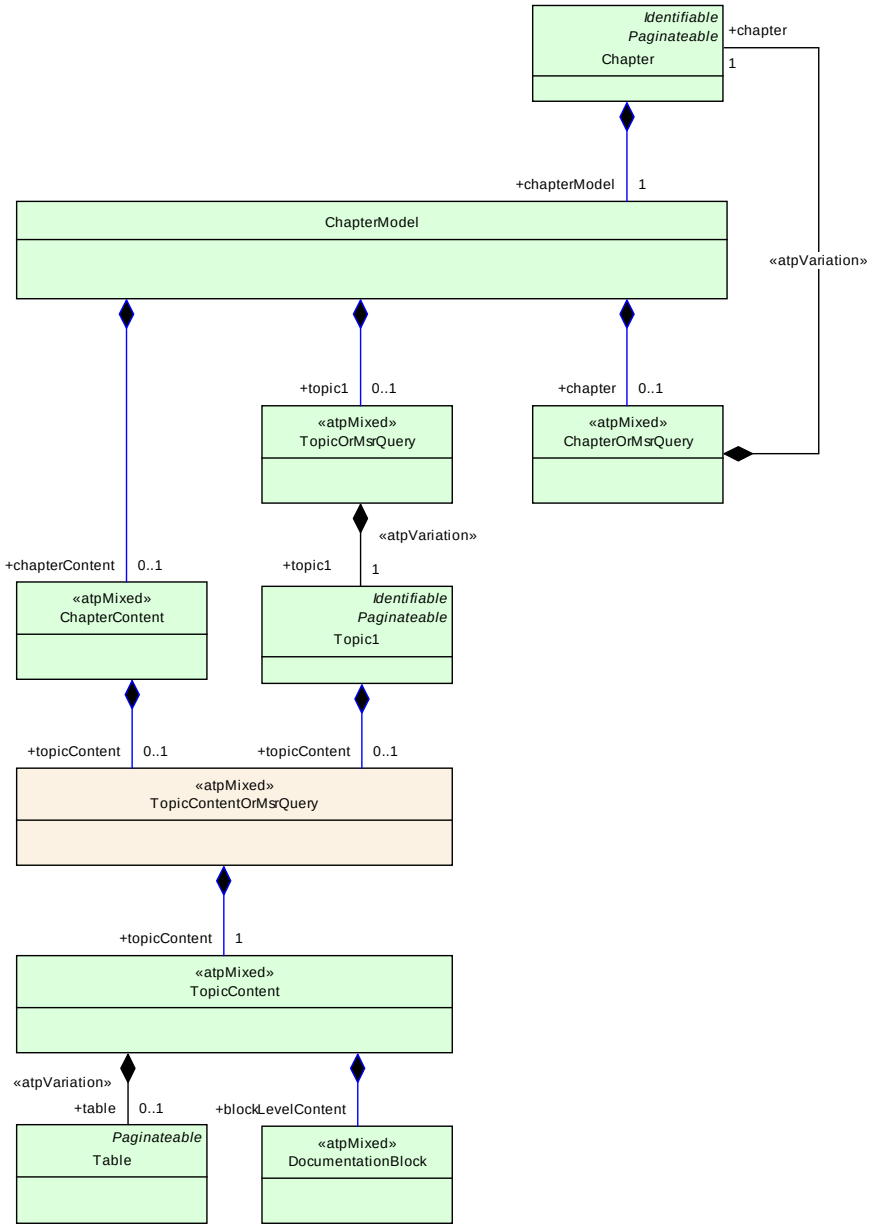

Figure 8.9: Standalone Documentation Overview

It is also provided as `Documentation` which is an `ArElement` of its own rights allowing for a reference to the document's context.

8.3.1 Documentation's Context

Figure 8.10 depicts how the documentation context is captured. AUTOSAR provides `Documentation`, which is a packageable element and can be used to depict documentation in the context of any identifiable element or even M1 instance.

[constr_2533] Documentation context is either a feature or an identifiable ⌈ One particular `DocumentationContext` shall be either a feature or an identifiable but not

both at the same time. If this is desired, one should create multiple `Documentation-Context`.⌋

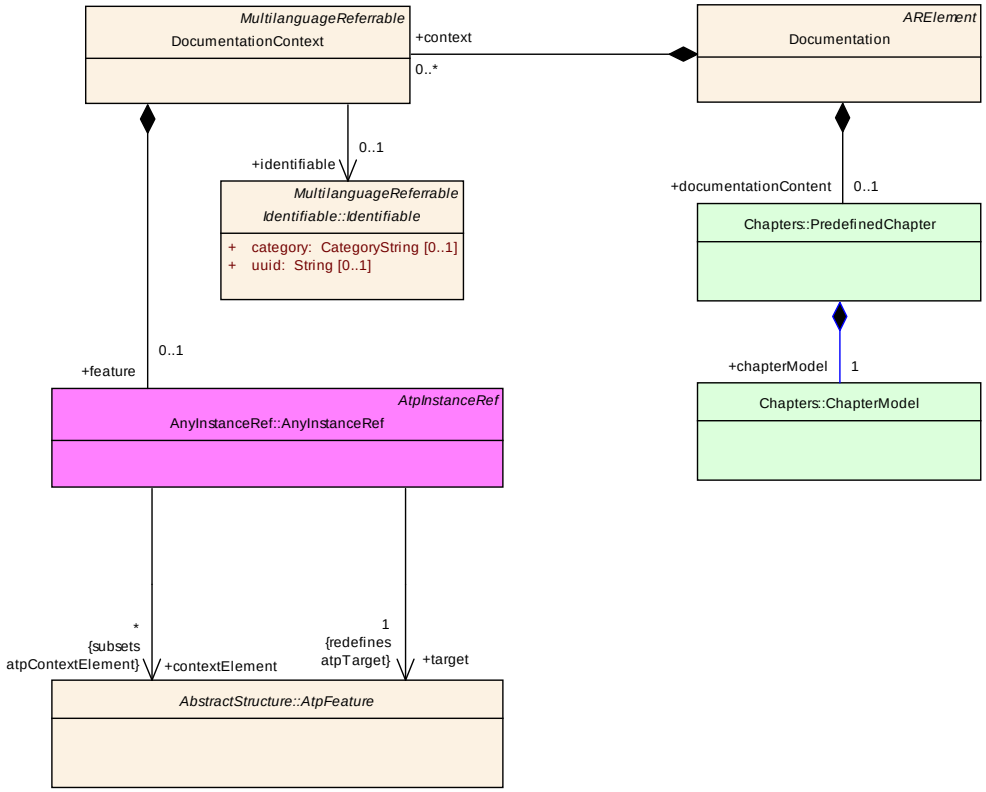

Figure 8.10: Standalone documentation as ArElement

Class	**DocumentationContext**			
Package	M2::AUTOSARTemplates::GenericStructure::DocumentationOnM1			
Note	This class represents the ability to denote a context of a so called standalone documentation. Note that this is an «atpMixed». The contents needs to be considered as ordered.			
Base	ARObject,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
feature	AtpFeature	0..1	iref	This referes to a paricular feature (instance in the M0 model) to which is the context of the documentation.
identifiable	Identifiable	0..1	ref	This is an identifiable object which is part of the context of the documentation.

Table 8.54: DocumentationContext

Class	**AnyInstanceRef**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::AnyInstanceRef			
Note	Describes a reference to any instance in an AUTOSAR model. This is the most generic form of an instance ref. Refer to the superclass notes for more details.			
Base	ARObject,AtpInstanceRef			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
base	AtpClassifier	1	ref	This is the base from which navigation path begins. **Stereotypes:** atpDerived
contextElement	AtpFeature	*	ref	This is one step in the navigation path specified by the instance ref.
target	AtpFeature	1	ref	This is the target of the instance ref.

Table 8.55: AnyInstanceRef

8.3.2 Chapter

The chapter element is composed of a title (`longName`), its content, one or more logical block grouped as a unit (`topic1` see chapter 8.3.4) and sub chapters. The chapter's content is composed of parameters, tables, and documentation blocks. The `PredefinedChapter` is a chapter which cannot be nested because it depicts a particular semantics. Anyhow it can have nested chapters inside.

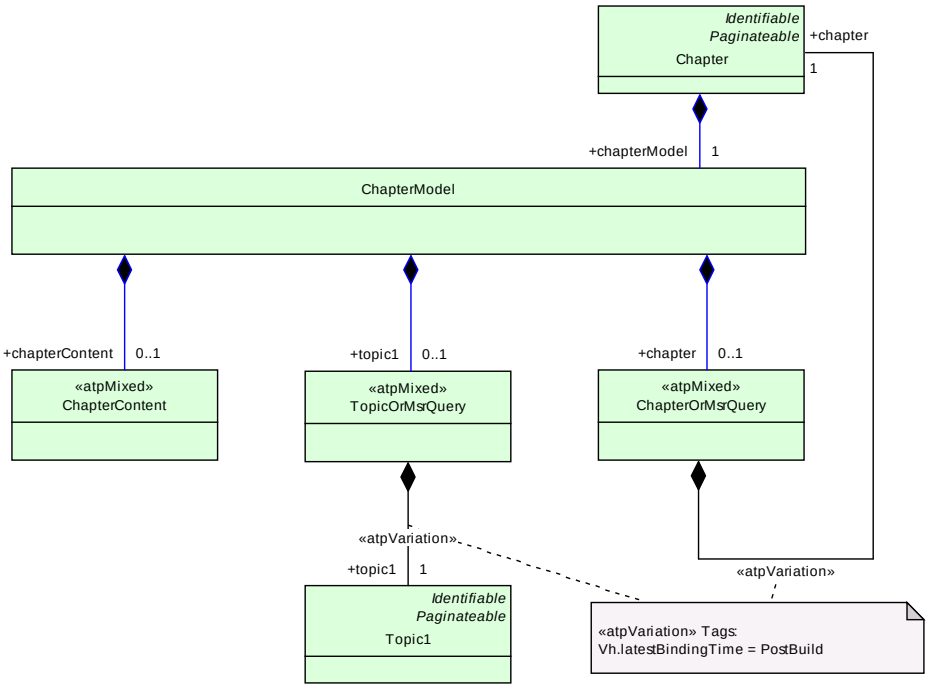

Figure 8.11: Chapter Overview

Class	**Chapter**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::Chapters			
Note	This meta-class represents a chapter of a document. Chapters are the primary structuring element in documentation.			
Base	ARObject,DocumentViewSelectable,Identifiable,MultilanguageReferrable,Paginateable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
chapterModel	ChapterModel	1	aggr	This represents the overall contents of the chapter. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.typeElement=false; xml.typeWrapperElement=false
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax must be defined by the applied help system respectively help system generator. Maybe it is a concatenated Identifier, but as of now we leave it as an arbitrary string. **Tags:** xml.attribute=true

Table 8.56: Chapter

Class	**ChapterModel**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::Chapters			
Note	This is the basic content model of a chapter except the Chapter title. This can be utilized in general chapters as well as in predefined chapters. A chapter has content on three levels: 1. chapter content 2. topics 3. subchapters			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
chapter	ChapterOrMsrQuery	0..1	aggr	This is a particular subchapter. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=200; xml.typeElement=false; xml.typeWrapperElement=false
chapterContent	ChapterContent	0..1	aggr	This is the chapter content which is not a topic or a subchapter. It is the content which is directly in the chapter. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=30; xml.typeElement=false; xml.typeWrapperElement=false

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
topic1	TopicOrMsrQuery	0..1	aggr	This is a topic within the chapter. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=170; xml.typeElement=false; xml.typeWrapperElement=false

Table 8.57: ChapterModel

Class	≪`atpMixed`≫ **ChapterContent**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::Chapters			
Note	This class represents the content which is directly in a chapter. It is basically the same as the one in a Topic but might have additional complex structures (e.g. Synopsis)			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
prms	Prms	1	aggr	This is a parameter table within a chapter. **Tags:** xml.sequenceOffset=150
topicContent	TopicContentOrMsrQuery	0..1	aggr	This is that part of a chapter content which may appear in a chapter as well as in a topic. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=40; xml.typeElement=false; xml.typeWrapperElement=false

Table 8.58: ChapterContent

Enumeration	**ChapterEnumBreak**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::PaginationAndView
Note	This allows to specify the page break policy of a paginatable element.
Literal	***Description***
break	This indicates the a page break shall be applied before the current block.
noBreak	This indicates that there is no need to force a page break before this block.

Table 8.59: ChapterEnumBreak

Class	**PredefinedChapter**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::Chapters			
Note	This represents a predefined chapter.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
chapterModel	ChapterModel	1	aggr	This is the content of the predefined chapter. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.typeElement=false; xml.typeWrapperElement=false

Attribute	Datatype	Mul.	Kind	Note

Table 8.60: PredefinedChapter

8.3.3 Tables in Documentation

AUTOSAR supports to use tables in documentation by the meta-class `Table` which is an implementation of the Oasis exchange table model ([17]). The model is depicted in Figure 8.12.

A table (`Table`) contains one or more partitions (`Tgroup`). The first partition has column specification (`Colspec`), which specifies the attributes of column within the partition.

Subsequent partitions can define their own column specification or inherit from the last partition that had a specification. A partition is composed of exactly one body (`Tbody`) one optional header (`Thead`) and one optional footer (`Tfoot`).

The body, header and footer are composed of one or more rows (`Row`). Each Row is composed of one or more entries (`Entry`), which contains a documentation block.

A table can also have a caption.

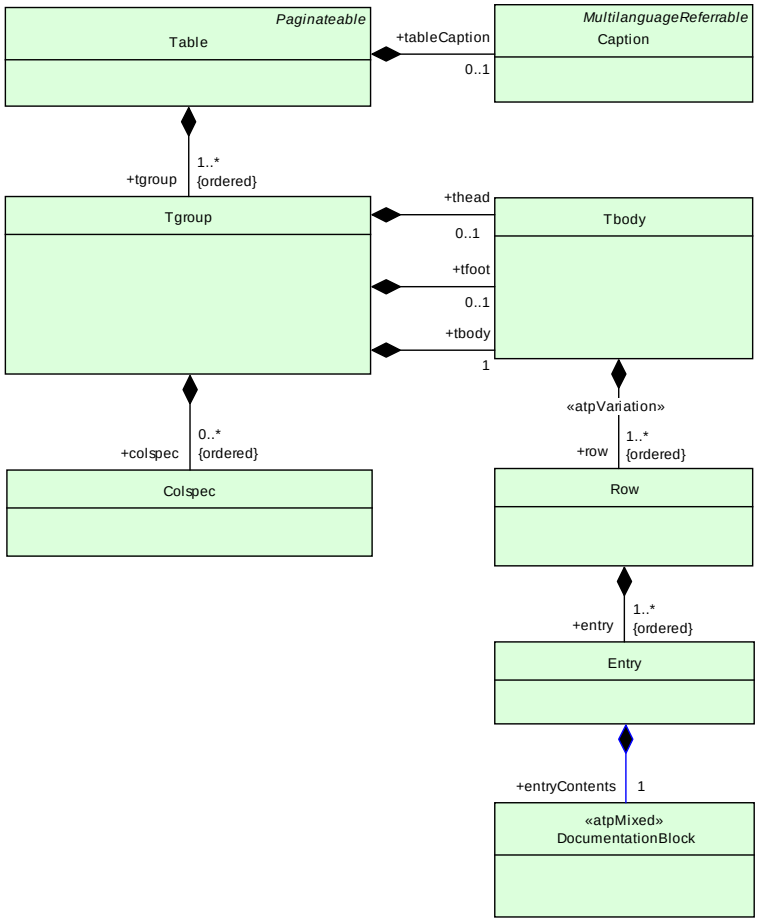

Figure 8.12: Table Model Overview

Class	**Table**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::OasisExchangeTable			
Note	This class implements an exchange table according to OASIS Technical Resolution TR 9503:1995.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
colsep	TableSeparator String	0..1	attr	Indicates if by default a line should be drawn between the columns of this table. **Tags:** xml.attribute=true
float	FloatEnum	1	attr	Indicate whether it is allowed to break the element. **Tags:** xml.attribute=true
frame	FrameEnum	0..1	attr	Used to defined the frame line around a table. **Tags:** xml.attribute=true
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax must be defined by the applied help system respectively help system generator. **Tags:** xml.attribute=true
orient	OrientEnum	0..1	attr	Indicate whether a table should be represented as landscape or portrait. • land : landscape • port : portrait **Tags:** xml.attribute=true
pgwide	NameToken	0..1	attr	Used to indicate wether the figure should take the complete page width (value = "pgwide") or not (value = "noPgwide"). **Tags:** xml.attribute=true
rowsep	TableSeparator String	0..1	attr	Indicates if by default a line should be drawn at the bottom of table rows. **Tags:** xml.attribute=true
tableCaption	Caption	0..1	aggr	This element specifies the table heading. **Tags:** xml.sequenceOffset=20
tabstyle	NameToken	0..1	attr	Indicates an external table style. **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
tgroup (ordered)	Tgroup	1..*	aggr	A table can be built of individual segments. Such a segment is called tgroup. **Tags:** xml.roleElement=true; xml.roleWrapperElement=false; xml.sequenceOffset=30; xml.typeElement=false; xml.typeWrapperElement=false

Table 8.61: Table

Enumeration	**FloatEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::OasisExchangeTable
Note	This enumerator specifies the policy how an objects floats on a page.
Literal	*Description*
float	This indicates that a page formatter is allowed to float the table to optimize the pagination. This is for example supported by TeX.
noFloat	This indicates that a page formatter is not allowed to float the object to optimize the pagination.

Table 8.62: FloatEnum

Enumeration	**FrameEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::OasisExchangeTable
Note	This enumerator specifies the policy, where to place a frame border around the table.
Literal	*Description*
all	Borders all around the table
bottom	Border at the bottom of the table
none	No borders around the table
sides	Borders at the sides of the table
top	Border at the top of the table
topbot	Borders at the top and bottom of the table

Table 8.63: FrameEnum

Class	**Tgroup**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::OasisExchangeTable			
Note	This meta-class represents the ability to denote a table section.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
align	AlignEnum	0..1	attr	Specifies how the cell entries shall be horizontally aligned within the specified TGROUP. Default is "LEFT" **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
cols	Integer	1	attr	This attribute represents the number of columns in the table. **Tags:** xml.attribute=true
colsep	TableSeparator String	0..1	attr	Indicates if by default a line shall be drawn between the columns of this table group. **Tags:** xml.attribute=true
colspec (ordered)	Colspec	*	aggr	This specifies one particular column specification in the table. There must be one entry for each column. **Tags:** xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false
rowsep	TableSeparator String	0..1	attr	Indicates if by default a line shall be drawn at the bottom of the rows in this table group. **Tags:** xml.attribute=true
tbody	Tbody	1	aggr	This is the main part of the table segment, called the table body. **Tags:** xml.sequenceOffset=60
tfoot	Tbody	0..1	aggr	This represents the footer of the table segement. This segment is printed at the end of the table or before a page break. **Tags:** xml.sequenceOffset=50
thead	Tbody	0..1	aggr	This represents the heading of the table section. The heading is usually repeated at the beginning of each new page. **Tags:** xml.sequenceOffset=40

Table 8.64: Tgroup

Enumeration	**AlignEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::OasisExchangeTable
Note	This enumerator specifies horizontal alignment.
Literal	***Description***
center	The content of the table is horizontally centered.
justify	This indicates that the content of table cell shall be justified (rendered as a block where white-space is expanded such that all lines are filled up).
left	This indicates that the content of a table cell is left justified.
right	This indicates that the content of a table cell is left justified.

Table 8.65: AlignEnum

Class	**Tbody**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::OasisExchangeTable			
Note	This meta-class represents a part within a table group. Such a part can be the table head, the table body or the table foot.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
row (ordered)	Row	1..*	aggr	This is a particular row in a table. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false
valign	ValignEnum	0..1	attr	Indicates how the cells in the rows shall be aligned. Default is inherited from tbody, otherwise it is "TOP" **Tags:** xml.attribute=true

Table 8.66: Tbody

Enumeration	**ValignEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::OasisExchangeTable
Note	This enumerator specifies vertical alignment.
Literal	***Description***
bottom	The contents of the table cell is bottom aligned.
middle	The contents of the table is vertically centered.
top	The contents of the table cell is top aligned.

Table 8.67: ValignEnum

Class	**Row**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::OasisExchangeTable			
Note	This meta-class represents the ability to express one row in a table.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
entry (ordered)	Entry	1..*	aggr	This represents one particular table cell. It is an entry in the table. **Tags:** xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false
rowsep	TableSeparator String	0..1	attr	Indicates if by default a line should be displayed below the row. **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
valign	ValignEnum	0..1	attr	Indicates how the cells in the rows shall be aligned. Default is inherited from tbody, otherwise it is "TOP" **Tags:** xml.attribute=true

Table 8.68: Row

Class	**Entry**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::OasisExchangeTable			
Note	This represents one particular table cell.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
align	AlignEnum	0..1	attr	Specifies how the cell ENTRY shall be horizontally aligned. Default is "LEFT" **Tags:** xml.attribute=true
colname	String	0..1	attr	Indicate the name of the column, where the entry should appear. **Tags:** xml.attribute=true
colsep	TableSeparator String	0..1	attr	Indicates whether a line should be displayed end of this entry. **Tags:** xml.attribute=true
entryContents	Documentation Block	1	aggr	This is the content of the TableEntry **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.typeElement=false; xml.typeWrapperElement=false
morerows	String	0..1	attr	Number of additional rows. Default is "0" **Tags:** xml.attribute=true
nameend	String	0..1	attr	When an entry spans multiple column this is the name of the last column. **Tags:** xml.attribute=true
namest	String	0..1	attr	When an entry spans multiple column this is the name of the first column. **Tags:** xml.attribute=true
rotate	String	0..1	attr	Indicates if the cellcontent shall be rotated. Default is 0; 1 would rotate the contents 90 degree counterclockwise. This attribute is defined by OASIS. **Tags:** xml.attribute=true

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
rowsep	TableSeparator String	0..1	attr	Indicates whether a line should be displayed at the bottom end of the cell. **Tags:** xml.attribute=true
spanname	String	0..1	attr	Capture the name of entry merging multiple columns. **Tags:** xml.attribute=true
valign	ValignEnum	0..1	attr	Indicates how the content of the cell shall be aligned. Default is inherited from row or tbody, otherwise "TOP" **Tags:** xml.attribute=true

Table 8.69: Entry

Primitive	**TableSeparatorString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::OasisExchangeTable
Note	This represents the ability to denote a separator string within an OASIS exchange table. • **0**: no line is displayed • **1**: line is displayed **Tags:** xml.xsd.customType=TABLE-SEPARATOR-STRING; xml.xsd.pattern=[0-1]; xml.xsd.type=string

Table 8.70: TableSeparatorString

8.3.4 Topics in Documentation

A topic (`Topic1`[6] is a logical unit, which is grouped and given a label (its `LongName`).

Class	**Topic1**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::Chapters			
Note	This meta-class represents a topic of a documentation. Topics are similar to chapters but they cannot be nested. They also do not appear in the table of content. Topics can be used to produce intermediate headlines thus structuring a chapter internally.			
Base	ARObject,DocumentViewSelectable,Identifiable,MultilanguageReferrable,Paginateable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

[6]The name topic1 is given to remain compatible with [16]

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax must be defined by the applied help system respectively help system generator. **Tags:** xml.attribute=true
topicContent	TopicContentOrMsrQuery	0..1	aggr	This is the content of the topic. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false

Table 8.71: Topic1

8.3.5 Parameter tables

Parameter tables can be used to collect numerical or textual parameters in a documentation. Such parameters should not to be confused with parameters in the software of an ECU. Parameter tables are intended to create a kind of data sheets.

Class	**Prms**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::GerneralParameters			
Note	This metaclass represents the ability to specify a parameter table. It can be used e.g. to specify parameter tables in a data sheet.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
label	MultilanguageLongName	0..1	aggr	This represents the caption of the parameter table. **Tags:** xml.sequenceOffset=20
prm	GeneralParameter	1..*	aggr	This represents one particular parameter in the the table. **Tags:** xml.roleElement=true; xml.roleWrapperElement=false; xml.sequenceOffset=30; xml.typeElement=false; xml.typeWrapperElement=false

Table 8.72: Prms

8.4 Document production

Production of e.g. printed documents is done using document processors. These processors determine the document content and layout. Nevertheless it is necessary to support this document production process by some policies in order to tweak the final output.

AUTOSAR provides basic support for a pagination policy (`Paginateable`) which allows to tweak the pages of generated documents.

Class	**Paginateable (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::PaginationAndView			
Note	This meta-class represents the ability to control the pagination policy when creating documents.			
Base	ARObject,DocumentViewSelectable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
break	ChapterEnumBreak	0..1	attr	This attributes allows to specify a forced page break. **Tags:** xml.attribute=true
keepWithPrevious	KeepWithPreviousEnum	0..1	attr	This attribute denotes the pagination policy. In particular it defines if the containing text block shall be kept together with the previous block. **Tags:** xml.attribute=true

Table 8.73: Paginateable

Enumeration	**ChapterEnumBreak**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::PaginationAndView
Note	This allows to specify the page break policy of a paginatable element.
Literal	***Description***
break	This indicates the a page break shall be applied before the current block.
noBreak	This indicates that there is no need to force a page break before this block.

Table 8.74: ChapterEnumBreak

Enumeration	**KeepWithPreviousEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::PaginationAndView
Note	This enumerator specifies a page break policy by controlling blocks which shall be kept together.
Literal	***Description***
keep	This indicates that the block shall be kept together with the previous block.
noKeep	This indicates that there is no need to keep the block with the previous one. This is the same as if the attribute itself is missing.

Table 8.75: KeepWithPreviousEnum

In addition to this, AUTOSAR provides support of multiple document views. Such views can on one hand be determined by retrieving particular meta-class instances from an M1 model. But in addition to this, document views are supported by `DocumentViewSelectable`.

Class	**DocumentViewSelectable (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::PaginationAndView			
Note	This meta-class represents the ability to be dedicated to a particular audience or document view.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
si	NameTokens	1	attr	This attribute allows to denote a semantic information which is used to identify documentation objects to be selected in customizable document views. It shall be defined in agreement between the involved parties. **Tags:** xml.attribute=true
view	ViewTokens	0..1	attr	This attribute lists the document views in which the object shall appear. If it is missing, the object appears in all document views. **Tags:** xml.attribute=true

Table 8.76: DocumentViewSelectable

Primitive	**ViewTokens**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::PaginationAndView
Note	This primitive specifies the tokens to specify a documentation view. **Tags:** xml.xsd.customType=VIEW-TOKENS; xml.xsd.pattern=(-?[a-zA-Z_]+)(( )+-?[a-zA-Z_]+)*; xml.xsd.type=string

Table 8.77: ViewTokens

An example for application of `view`

```
<P VIEW="-MYVIEW">  ... always there
    except explicitly created MYVIEW ...
<P VIEW="MYVIEW"> ... only printed
    in MYVIEW document ...
<P VIEW="MYVIEW YOURVIEW"> ... only printed
    in MYVIEW and in YOURVIEW Document...
```

8.5 Including generated documentation parts

AUTOSAR supports an approach where parts of the documentation are automatically generated and included at a particular location within the documentation. This support is provided by the so called MsrQuery class. This class allows to represent the properties as well as the result of the inclusion. This allows to visualize and exchange the intermediate result after the generated parts were included. By this it is not necessary that all involved parties are able to perform the inclusion process.

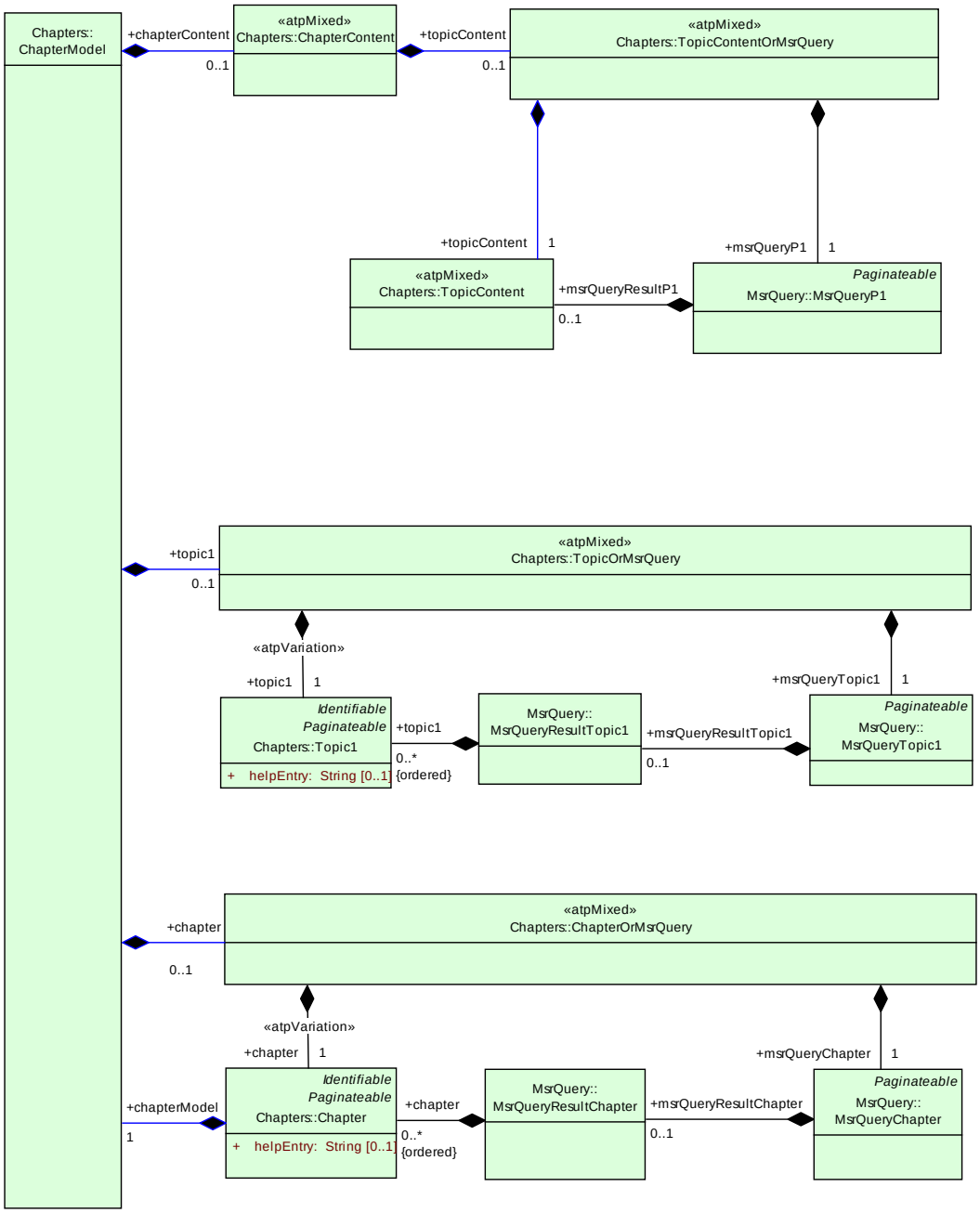

Figure 8.13: Including generated documentation parts by MsrQuery

The following meta-classes represent the alternative of manually edited documentation and included generated parts.

Class	**≪`atpMixed`≫ TopicContentOrMsrQuery**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::Chapters			
Note	This meta-class represents a topic or a topic content which is generated using queries.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
msrQuery P1	MsrQueryP1	1	aggr	This represents automatically contributed contents provided by an msrquery.
topicConte nt	TopicContent	1	aggr	This is the content of a topic. **Tags:** xml.roleElement=false

Table 8.78: TopicContentOrMsrQuery

Class	≪`atpMixed`≫ **TopicOrMsrQuery**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::Chapters			
Note	This class provides the alternative of a Topic with an MsrQuery which delivers a topic.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
msrQueryT opic1	MsrQueryTopic 1	1	aggr	This represents automatically contributed topics provided by an msrquery. **Tags:** xml.sequenceOffset=190
topic1	Topic1	1	aggr	This is used to create particcular topics within a chapter. A topic is similar to a subchapter, but cannot be nesxted and will not appear in the table of contents of the document. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.sequenceOffset=180

Table 8.79: TopicOrMsrQuery

Class	≪`atpMixed`≫ **ChapterOrMsrQuery**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::Chapters			
Note	This meta-class represents the ability to denote a particular chapter or a query returning a chapter.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
chapter	Chapter	1	aggr	This establishes a subschapter. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.sequenceOffset=210
msrQuery Chapter	MsrQueryChapt er	1	aggr	This represents automatically contributed chapters provided by an msrquery. **Tags:** xml.sequenceOffset=220

Table 8.80: ChapterOrMsrQuery

The following meta-classes represent the included generated parts.

Class	**MsrQueryP1**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::MsrQuery			
Note	This meta-class represents the ability to express a query wihich yelds the content of a topic as a result.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
msrQuery Props	MsrQueryProps	1	aggr	This is argument and properties of the paragraph query. **Tags:** xml.sequenceOffset=20
msrQuery ResultP1	TopicContent	0..1	aggr	This represents the result of the query. **Tags:** xml.sequenceOffset=30

Table 8.81: MsrQueryP1

Class	**MsrQueryTopic1**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::MsrQuery			
Note	This meta-class represents the ability to specify a query which yields a set of topics as a result.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
msrQuery Props	MsrQueryProps	1	aggr	This is argument and properties of the topic query. **Tags:** xml.sequenceOffset=20
msrQuery ResultTopic1	MsrQueryResultTopic1	0..1	aggr	This represents the result of the query. **Tags:** xml.sequenceOffset=30

Table 8.82: MsrQueryTopic1

Class	**MsrQueryChapter**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::MsrQuery			
Note	This meta-class represents the ability to express a query which yields a set of chapters as a result.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
msrQuery Props	MsrQueryProps	1	aggr	This is argument and properties of the chapter query. **Tags:** xml.sequenceOffset=20
msrQuery ResultChapter	MsrQueryResultChapter	0..1	aggr	This represents the result of the query. **Tags:** xml.sequenceOffset=30

Table 8.83: MsrQueryChapter

The following meta-classes control the inclusion process.

Class	**MsrQueryProps**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::MsrQuery			
Note	This metaclass represents the ability to specificy a query which yields some documentation text. The qualities of the result are determined by the context in which the query is used.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
comment	String	0..1	attr	This element contains a commentary in text form. **Tags:** xml.sequenceOffset=40
msrQuery Arg	MsrQueryArg	*	aggr	This element specifies an argument within an MsrQuery. **Tags:** xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=30; xml.type Element=false; xml.typeWrapperElement=false
msrQuery Name	String	1	attr	This element specifies the name of the MSR-QUERY triggered. **Tags:** xml.sequenceOffset=20

Table 8.84: MsrQueryProps

Class	**MsrQueryArg**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::MsrQuery			
Note				
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
arg	String	1	attr	This is the value of the argument. **Tags:** xml.roleElement=false; xml.roleWrapper Element=false; xml.typeElement=false; xml.type WrapperElement=false
si	NameToken	1	attr	This denotes the name of the query argument (semantic information) **Tags:** xml.attribute=true

Table 8.85: MsrQueryArg

The following meta-classes represent the included results.

Class	**MsrQueryResultChapter**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::MsrQuery			
Note	This metaclass represents the result of an msrquery which is a set of chapters.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
chapter (ordered)	Chapter	*	aggr	This is one particular chapter in the query result. **Tags:** xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false

Table 8.86: MsrQueryResultChapter

Class	**MsrQueryResultTopic1**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::MsrQuery			
Note	This metaclass represents the ability to express the result of a query which is a set of topics.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
topic1 (ordered)	Topic1	*	aggr	This represents one particular topic in the query result. **Tags:** xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false

Table 8.87: MsrQueryResultTopic1

8.6 Handling Multiple Languages in an AUTOSAR Artifact

AUTOSAR supports a multi-language documentation[7], where each construct of the document can have the optional attribute L with as value the language in which it is written. The languages available in a document are defined in its `adminData` (see 3 for details).

The following block level elements and their sub-constructs support multilingual text/-graphics:

- paragraph
- formula
- figure
- verbatim

[7]based on [16]

Figure 8.14 illustrates the approach for multilanguage support using LongName as an example. It shows how the single language and the multilanguage version are derived from a common model.

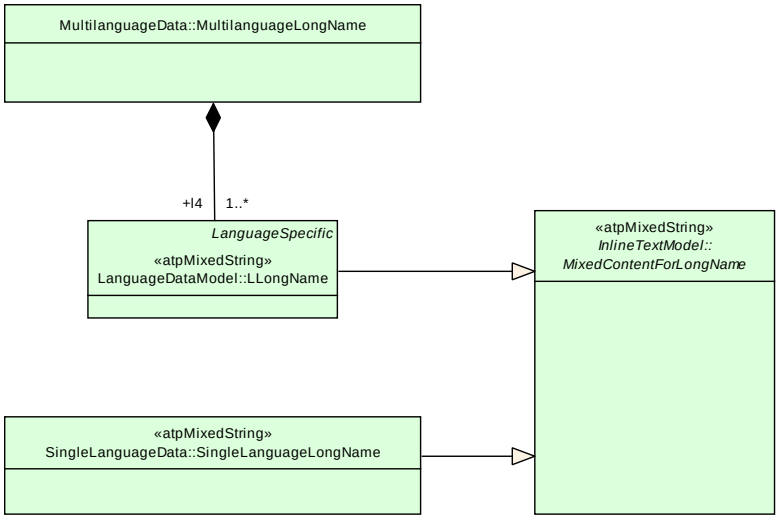

Figure 8.14: The `MultilanguageLongname`

The annotation of each construct with the language it which it is written allows the author of a documentation block to write interlinear it in two or more languages weaved together in a single source document, then to generate the documentation block in each of these languages.

[constr_2523] Used languages need to be consistent ⌈ The used languages of an AUTOSAR file are specified in the top level `adminData` as shown in Chapter 3. All other elements shall be provided in the languages specified for the document. ⌋

This approach supports a better maintainability. In a documentation block written in English, French and German, for example, it is easier to maintain the three versions consistent when each paragraph is immediately followed by its translation in the other languages as when they are found at completely different locations or worse separated documents.

The following example illustrate this approach:

Listing 8.1: Example of Excerpt from a Multilingual Documentation Block in ARXML

```
<AR-PACKAGE>
  <SHORT-NAME>Documentation</SHORT-NAME>
  <INTRODUCTION>
    <P>
      <L-1 L="EN">
        In a documentation block written  in English,
        French and German, for example, it is easier
        to maintain the three versions consistent
        when each paragraph is immediately followed
        by its translation in the other languages as
        when they are found at completely different
        locations or worse separated documents.
        </L-1>
```

```
        <L-1 L="FR">
          Dans un bloque documentaire écrit en anglais,
          français et allemand, par exemple, il est plus
          facile de maintenir les trois versions
          consistantes entre elles, quand chaque paragraphe
          est suivi immédiatement de sa traduction dans les
          autres langues que si ces dernières ce trouvaient
          dans des endroits différents ou même pire dans des
          documents séparés.
          </L-1>
      </P>
    </INTRODUCTION>
  </AR-PACKAGE>
```

Class	**MultiLanguageVerbatim**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::MultilanguageData			
Note	This class represents multilingual Verbatim. Verbatim means, that white-space is maintained. When Verbatim is rendered in PDF or Online media, white-space is obeyed. Blanks are rendered as well as newline characters.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
allowBreak	NameToken	0..1	attr	This indicates if the verbatim text might be split on multiple pages. Default is "1". **Tags:** xml.attribute=true
float	FloatEnum	0..1	attr	Indicate whether it is allowed to break the element. The following values are allowed: **Tags:** xml.attribute=true
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax must be defined by the applied help system respectively help system generator. **Tags:** xml.attribute=true
l5	LVerbatim	1..*	aggr	This the text in one particular language. **Tags:** xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false
pgwide	PgwideEnum	0..1	attr	Used to indicate wether the figure should take the complete page width (value = "pgwide") or not (value = "noPgwide"). **Tags:** xml.attribute=true

Table 8.88: MultiLanguageVerbatim

Class	≪`atpMixedString`≫ **LVerbatim**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::LanguageDataModel			
Note	MixedContentForVerbatim in one particular language. The language is denoted in the attribute l.			
Base	ARObject,LanguageSpecific,MixedContentForVerbatim,WhitespaceControlled			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 8.89: LVerbatim

Class	**MultiLanguageOverviewParagraph**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::MultilanguageData			
Note	This is the content of a multilingual paragraph in an overview item.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
l2	LOverviewParagraph	1..*	aggr	This represents the text in one particular language. **Tags:** xml.roleElement=true; xml.roleWrapperElement=false; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false

Table 8.90: MultiLanguageOverviewParagraph

Class	≪`atpMixedString`≫ **LOverviewParagraph**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::LanguageDataModel			
Note	MixedContentForOverviewParagraph in one particular language. The language is denoted in the attribute l.			
Base	ARObject,LanguageSpecific,MixedContentForOverviewParagraph			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 8.91: LOverviewParagraph

Class	**MultiLanguageParagraph**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::MultilanguageData			
Note	This is the content model of a multilingual paragraph in a documentation.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	Datatype	Mul.	Kind	Note
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax must be defined by the applied help system respectively help system generator. **Tags:** xml.attribute=true
l1	LParagraph	1..*	aggr	This is the paragraph content in one partiucular language. **Tags:** xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false

Table 8.92: MultiLanguageParagraph

Class	≪`atpMixedString`≫ **LParagraph**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::LanguageDataModel			
Note	This is the text for a paragraph in one particular language. The language is denoted in the attribute l.			
Base	ARObject,LanguageSpecific,MixedContentForParagraph			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 8.93: LParagraph

Class	MultiLanguagePlainText			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::MultilanguageData			
Note	This is a multilingual plaint Text.It is intended to be rendered as a paragraph.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
l10	LPlainText	1..*	aggr	This is the plain text in one particular language. **Tags:** xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false

Table 8.94: MultiLanguagePlainText

Class	LPlainText			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::LanguageDataModel			
Note	MixedContentForPlainText in one particular language. The language is denoted in the attribute l.			
Base	ARObject,LanguageSpecific,MixedContentForPlainText,WhitespaceControlled			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Table 8.95: LPlainText

Class	**LanguageSpecific (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::TextModel::LanguageDataModel			
Note	This meta-class represents the ability to denote a particular language for which an object is applicable.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
l	LEnum	1	attr	'This attribute denotes the language in which the language specific document entity is given. Note that "FOR-ALL" means, that the entity is applicable to all languages. It is language neutral. It follows ISO 639-1:2002 and is specified in upper case. **Tags:** xml.attribute=true; xml.enforceMin Multiplicity=true

Table 8.96: LanguageSpecific

A Glossary

Artifact This is a Work Product Definition that provides a description and definition for tangible work product types. Artifacts may be composed of other artifacts ([18]).

At a high level, an artifact is represented as a single conceptual file.

AUTOSAR Tool This is a software tool which supports one or more tasks defined as AUTOSAR tasks in the methodology. Depending on the supported tasks, an AUTOSAR tool can act as an authoring tool, a converter tool, a processor tool or as a combination of those (see separate definitions).

AUTOSAR Authoring Tool An AUTOSAR Tool used to create and modify AUTOSAR XML Descriptions. Example: System Description Editor.

AUTOSAR Converter Tool An AUTOSAR Tool used to create AUTOSAR XML files by converting information from other AUTOSAR XML files. Example: ECU Flattener

AUTOSAR Definition This is the definition of parameters which can have values. One could say that the parameter values are Instances of the definitions. But in the meta model hierarchy of AUTOSAR, definitions are also instances of the meta model and therefore considered as a description. Examples for AUTOSAR definitions are: `EcucParameterDef`, `PostBuildVariantCriterion`, `SwSystemconst`.

AUTOSAR XML Description In AUTOSAR this means "filled Template". In fact an AUTOSAR XML description is the XML representation of an AUTOSAR model.

The AUTOSAR XML description can consist of several files. Each individual file represents an AUTOSAR partial model and shall validate successfully against the AUTOSAR XML schema.

AUTOSAR Meta-Model This is an UML2.0 model that defines the language for describing AUTOSAR systems. The AUTOSAR meta-model is an UML representation of the AUTOSAR templates. UML2.0 class diagrams are used to describe the attributes and their interrelationships. Stereotypes, UML tags and OCL expressions (object constraint language) are used for defining specific semantics and constraints.

AUTOSAR Model This is a representation of an AUTOSAR product. The AUTOSAR model represents aspects suitable to the intended use according to the AUTOSAR methodology.

Strictly speaking, this 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.

AUTOSAR Partial Model In AUTOSAR, the possible partitioning of models is marked in the meta-model by ≪`atpSplitable`≫. One partial model is represented in an AUTOSAR XML description by one file. The partial model does not need to fulfill all semantic constraints applicable to an AUTOSAR model.

AUTOSAR Processor Tool An AUTOSAR Tool used to create non-AUTOSAR files by processing information from AUTOSAR XML files. Example: RTE Generator

AUTOSAR Template The term "Template" is used in AUTOSAR to describe the format different kinds of descriptions. The term template comes from the idea, that AUTOSAR defines a kind of form which shall be filled out in order to describe a model. The filled form is then called the description.

In fact the AUTOSAR templates are now defined as a meta model.

AUTOSAR XML Schema This is a W3C XML schema that defines the language for exchanging AUTOSAR models. This Schema is derived from the AUTOSAR meta model. The AUTOSAR XML Schema defines the AUTOSAR data exchange format.

Blueprint This is a model from which other models can be derived by copy and refinement. Note that in contrast to meta model resp. types, this process is *not* an instantiation.

Instance Generally this is a particular exemplar of a model or of a type.

Meta-Model This defines the building blocks of a model. In that sense, a Meta-Model represents the language for building models.

Meta-Data This includes pertinent information about data, including information about the authorship, versioning, access-rights, timestamps etc.

Model A Model is an simplified representation of reality. The model represents the aspects suitable for an intended purpose.

Partial Model This is a part of a model which is intended to be persisted in one particular artifact.

Pattern in GST : This is an approach to simplify the definition of the meta model by applying a model transformation. This transformation crates an enhanced model out of an annotated model.

Property A property is a structural feature of an object. As an example a “connector” has the properties “receive port” and “send port”

Properties are made variant by the ≪atpVariation≫.

Prototype This is the implementation of a role of a type within the definition of another type. In other words a type may contain Prototypes that in turn are typed by "Types". Each one of these prototypes becomes an instance when this type is instantiated.

Type A type provides features that can appear in various roles of this type.

Value This is a particular value assigned to a “Definition”.

Variability Variability of a system is its quality to describe a set of variants. These variants are characterized by variant specific property settings and / or selections.

As an example, such a system property selection manifests itself in a particular “receive port” for a connection.

This is implemented using the ≪atpVariation≫.

Variant A system variant is a concrete realization of a system, so that all its properties have been set respectively selected. The software system has no variability anymore with respect to the binding time.

This is implemented using `EvaluatedVariantSet`.

Variation Binding A variant is the result of a variation binding process that resolves the variability of the system by assigning particular values/selections to all the system’s properties.

This is implemented by `VariationPoint`.

Variation Binding Time The variation binding time determines the step in the methodology at which the variability given by a set of variable properties is resolved.

This is implementing by `vh.LatestBindingtime` at the related properties .

Variation Definition Time The variation definition time determines the step in the methodology at which the variation points are defined.

Variation Point A variation point indicates that a property is subject to variation. Furthermore, it is associated with a condition and a binding time which define the system context for the selection / setting of a concrete variant.

This is implemented by `VariationPoint`.

B Constraint History

B.1 Constraint History R4.0.1

B.1.1 Added Constraints

Number	Heading
[constr_2501]	Blueprint of blueprints are not supported
[constr_2502]	Merged model must be compliant to the meta-model.
[constr_2503]	Bound model must be compliant to the meta model
[constr_2504]	Constraint to latest binding time
[constr_2505]	Multiplicity after binding
[constr_2506]	Attributes in property set pattern
[constr_2507]	`EvaluatedVariantSet`
[constr_2508]	`shortName`
[constr_2509]	`ReferenceBase`
[constr_2510]	only one default `ReferenceBase`
[constr_2511]	Named reference bases must be available
[constr_2512]	`shortName` uniqueness constraint for variants
[constr_2513]	splitted variants must have a `shortLabel`
[constr_2514]	`shortLabel` in `VariationPoint` must be unique
[constr_2515]	Avoid conflicting package categories
[constr_2516]	Return type of Formula
[constr_2517]	`postbuildVariantCondition` only for PostBuild
[constr_2518]	Binding time is constrained
[constr_2519]	`PredefinedVariants` need to be consistent
[constr_2520]	Nesting of lists shall be limited
[constr_2521]	The `shortLabel` in `VariationPoint` must be unique
[constr_2522]	Notes should not be nested
[constr_2523]	Used languages need to be consistent
[constr_2524]	Non splitable elements in one file
[constr_2525]	Non splitable elements shall not be repeated
[constr_2530]	`InstanceRefs` must be consistent
[constr_2531]	`AtpInstanceRef` must be close to the base
[constr_2533]	Documentation context is either a feature or an identifiable
[constr_2534]	Limits of unlimited Integer

Table B.1: Added Constraints in 4.0.1

B.2 Constraint History R4.0.2

B.2.1 Added Constraints

Number	Heading
[constr_2537]	Variation of packagable element is limited to components resp. modules
[constr_2538]	Global reference is limited to certain elements

Table B.2: Added Constraints in 4.0.2

B.2.2 changed Constraints

Number	Heading
[constr_2511]	Named reference bases must be available
[constr_2519]	`PredefinedVariants` need to be consistent

Table B.3: changed Constraints in 4.0.2

B.3 Constraint History R4.0.3

B.3.1 Added Constraints

Number	Heading
[constr_2547]	ordered collections cannot be split into partial models
[constr_2557]	no `VariationPoint`s with `latestBindingTime` set to `BlueprintDerivation` in system configurations
[constr_2558]	Only `blueprintCondition/blueprintValue` if `Vh.latestBindingTime` is `BlueprintDerivationTime`
[constr_2559]	no nested `VariationPoint`
[constr_4055]	ICS may not contain blueprints

Table B.4: Added Constraints in 4.0.3

B.3.2 changed Constraints

Number	Heading
[constr_2530]	`InstanceRefs` must be consistent
[constr_2508]	Name space of `shortName`

Table B.5: changed Constraints in 4.0.3