Document Title	Standardization Template
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	535
Document Classification	Standard

Document Version	1.0.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History			
Date	**Version**	**Changed by**	**Description**
30.10.2011	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] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate.pdf

[2] Requirements on Standardization Template
AUTOSAR_RS_StandardizationTemplate.pdf

[3] Key words for use in RFCs to Indicate Requirement Levels
http://www.ietf.org/rfc/rfc2119.txt

[4] Predefined Names
AUTOSAR_TR_PredefinedNames.pdf

[5] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[6] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate.pdf

[7] ANTLR parser generator V3

[8] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[9] Unique Names for Documentation, Measurement and Calibration: Modeling and Naming Aspects including Automatic Generation
AUTOSAR_TR_AIMeasurementCalibrationDiagnostics

[10] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate.pdf

[11] SW-C and System Modeling Guide
AUTOSAR_TR_SWCModelingGuide.pdf

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

1 Introduction

AUTOSAR models are in many cases not created from scratch but existing content is taken as the basis. The existing content could be contributed by the AUTOSAR initiative itself in form of standardized model elements.

This document specifies the Standardization Template. This template is intended to support the delivery of standardized model elements by AUTOSAR and others.

AUTOSAR 4.0 already specifies the blueprint approach for standardization. This approach is continued and refined by the Standardization Template. It thereby replaces Appendix A in Software Component Template ([1]).

As an particular example, let us consider the standardization of application interfaces. That is, in terms of the AUTOSAR meta-model the standardization mainly applies to the definition of `PortPrototype`s for specific purposes.

Due to the structure of the AUTOSAR meta-model it is not possible to merely express a standardized `PortPrototype` because for good reasons the latter does not exist on its own but is always owned by a `SwComponentType`.

The Standardization Template specifies the approach to overcome this situation.

For more details such as use cases please refer to [2].

1.1 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.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as follows, based on [3].

Note that the requirement level of the document in which they are used modifies the force of these words.

- MUST: This word, or the adjective "LEGALLY REQUIRED", means that the definition is an absolute requirement of the specification due to legal issues.
- MUST NOT: This phrase, or the phrase "MUST NOT", means that the definition is an absolute prohibition of the specification due to legal issues.
- SHALL: This phrase, or the adjective "REQUIRED", means that the definition is an absolute requirement of the specification.
- SHALL NOT: This phrase means that the definition is an absolute prohibition of the specification.
- SHOULD: This word, or the adjective "RECOMMENDED", means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED", means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item.

An implementation, which does not include a particular option, SHALL be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, SHALL be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

1.2 Requirements Tracing

The following table references the requirements specified in [2] and links to the fulfillments of these.

Requirement	Description	Satisfied by
[RS_STDT_0001]	Shall support and explain Blueprints in general	[TPS_STDT_0002] [TPS_STDT_0027] [TPS_STDT_0042]
[RS_STDT_0002]	Formalized description of BSW SWS	[TPS_STDT_0014] [TPS_STDT_0040] [TPS_STDT_0041] [TPS_STDT_0049]
[RS_STDT_0003]	Shall allow to represent port blueprints	[TPS_STDT_0007] [TPS_STDT_0047]
[RS_STDT_0004]	Shall allow to represent name patterns	[TPS_STDT_0003] [TPS_STDT_0047] [TPS_STDT_0055]
[RS_STDT_0005]	Shall support keywords and keyword abbreviations	[TPS_STDT_0004] [TPS_STDT_0012]
[RS_STDT_0006]	Shall be implemented without compatibility problems to existing template	[TPS_STDT_0033] [TPS_STDT_0041] [TPS_STDT_0047]
[RS_STDT_0007]	Shall be based on the AUTOSAR schema	[TPS_STDT_0033] [TPS_STDT_0041] [TPS_STDT_0047]
[RS_STDT_0008]	Shall provide means to support analyzing the conformity of implementations with the AUTOSAR standards	[TPS_STDT_0001] [TPS_STDT_0003] [TPS_STDT_0012] [TPS_STDT_0042] [TPS_STDT_0048] [TPS_STDT_0052]
[RS_STDT_0009]	Shall be able to represent requirements stated in SWS	[TPS_STDT_0001] [TPS_STDT_0042] [TPS_STDT_0050] [TPS_STDT_0052]
[RS_STDT_0010]	Shall refer to ECUC parameter definition	[TPS_STDT_0025] [TPS_STDT_0040]
[RS_STDT_0011]	Shall be able to standardize components	[TPS_STDT_0024]
[RS_STDT_0012]	Shall be able to standardize architecture	[TPS_STDT_0024]
[RS_STDT_0013]	Shall be able to express parts of reference paths resp. package hierarchies	[TPS_STDT_0013] [TPS_STDT_0051]
[RS_STDT_0014]	Shall be able to express levels of obligation	[TPS_STDT_0028]
[RS_STDT_0015]	Shall support different Approaches to derive from Blueprints	[TPS_STDT_0028]
[RS_STDT_0016]	Shall be able to express information about the state of model elements	[TPS_STDT_0038]
[RS_STDT_0017]	Shall cover the compatibility of blueprints and derived objects	[TPS_STDT_0005] [TPS_STDT_0008] [TPS_STDT_0051]
[RS_STDT_0018]	Shall allow to describe the dependencies of APIs (e.g. invocation and callback/polling interfaces)	[TPS_STDT_0014] [TPS_STDT_0048]
[RS_STDT_0019]	Shall define the mandatory semantics for a Blueprint	[TPS_STDT_0003] [TPS_STDT_0028] [TPS_STDT_0048]

Requirement	Description	Satisfied by
[RS_STDT_0020]	Shall support variants of a VariableDataprototype	[TPS_STDT_0028] [TPS_STDT_0030] [TPS_STDT_0044] [TPS_STDT_0045] [TPS_STDT_0046]
[RS_STDT_0021]	Shall support multiple instantiation for an example SWC with PortBlueprint	[TPS_STDT_0003] [TPS_STDT_0036] [TPS_STDT_0037]
[RS_STDT_0022]	Means of exchange format between stakeholders for blueprints	[TPS_STDT_0025]
[RS_STDT_0023]	Shall be able to standardize Alias Names	[TPS_STDT_0011]
[RS_STDT_0024]	Shall be able to standardize Unique Names and Display Names	[TPS_STDT_0031]
[RS_STDT_0025]	Shall be able to standardize life cycle states	[TPS_STDT_0043]
[RS_STDT_0026]	Shall allow to represent port interface blueprints	[TPS_STDT_0009]
[RS_STDT_0027]	Shall allow to evaluate the integrity of Blueprints	[TPS_STDT_0034]
[RS_STDT_0028]	Shall allow to generate BSW ”Standard AUTOSAR Interface” description from model	[TPS_STDT_0023]
[RS_STDT_0029]	Shall be able to represent further Blueprints	[TPS_STDT_0014] [TPS_STDT_0015] [TPS_STDT_0016] [TPS_STDT_0017] [TPS_STDT_0018] [TPS_STDT_0019] [TPS_STDT_0020] [TPS_STDT_0022] [TPS_STDT_0023] [TPS_STDT_0035] [TPS_STDT_0049]
[RS_STDT_0030]	Shall allow to standardize package structures	[TPS_STDT_0013]

2 Support for Traceability

AUTOSAR has defined five levels of requirements for its standardization work:

1. AUTOSAR Project Objectives
2. AUTOSAR Main Requirements
3. AUTOSAR Features on BSW & RTE, Methodolgy & Templates, Application Interfaces
4. AUTOSAR Requirements Specifications (RS, SRS)
5. AUTOSAR Specifications (SWS, TPS, AI, TR, MOD, EXP etc.)

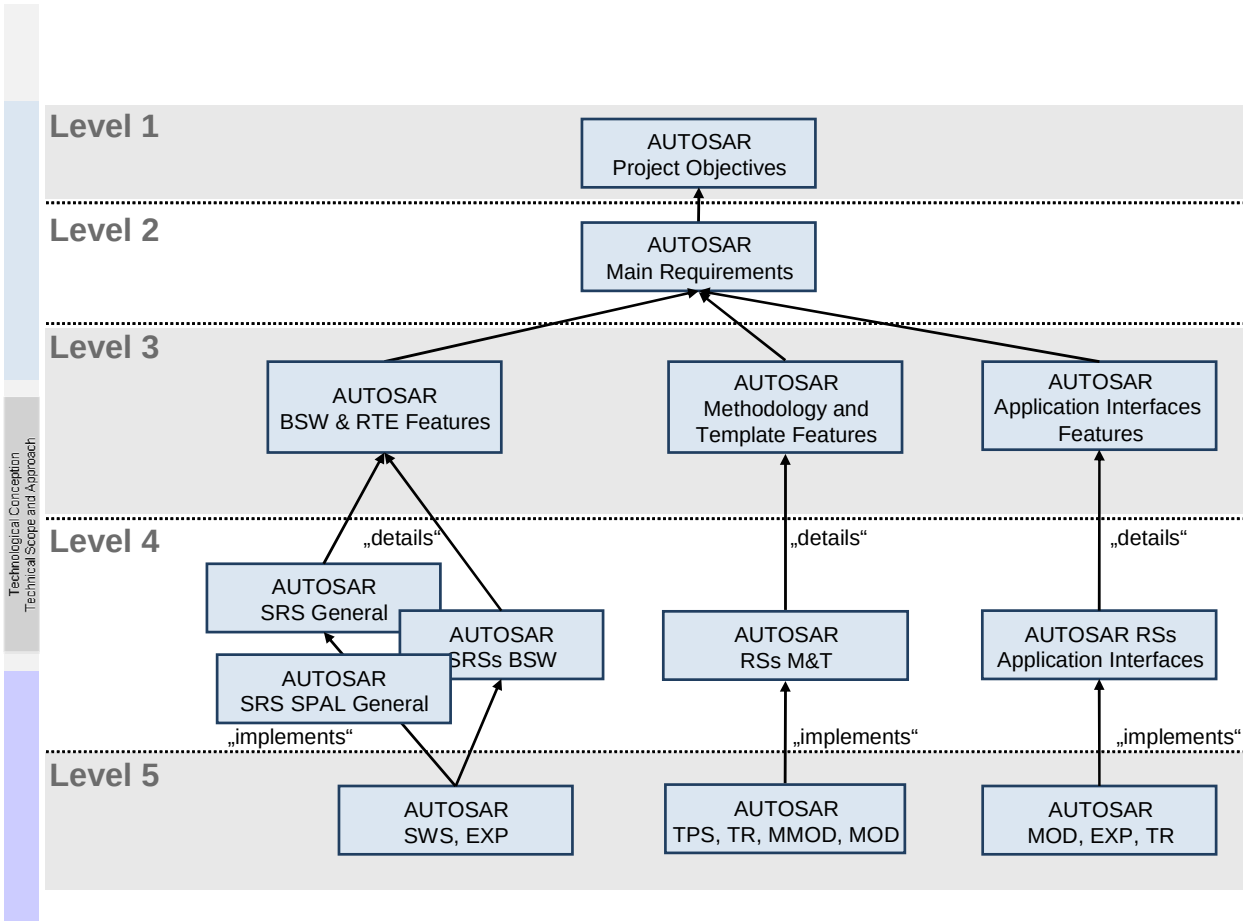

Figure 2.1: Specification levels

[TPS_STDT_0001] Support bottom up tracing ⌈ Standardization Template supports bottom up tracing between these levels by the meta-class `Traceable`. This allows to represent traceable entities and to establish traces between those. These entities reside within a `DocumentationBlock`. One prominent place is `DocumentationBlock.trace` in particular within `Identifiable.introduction`. ⌋*(RS_STDT_0008, RS_STDT_0009)*

`Traceable` is specialized in

- `TraceableText`: This represents a paragraph level text which can be referenced in order to establish requirements tracing. It is an abstract class from which particular specializations support specific kinds of tracing such as requirements / constraints.

  [constr_2540] Tagged text category ⌈ The `category` of `TraceableText` shall be one of

  `SPECIFICATION_ITEM` The text represents a particular item in the specification. Such an item is a requirement for the implementation of the software specification.

  `REQUIREMENT_ITEM` The text represents a particular requirement. Such an item is applicable primarily in requirement specifications.

  `CONSTRAINT_ITEM` The text represents a particular constraint. Such an item is applicable primarily in template specifications. It is similar to a specification item but represents issues that may be validated automatically e.g. by a tool.

  `IMPLEMENTATION_ITEM` The text represents a short description of an implementation. It is applicable primarily within the `introduction` of a model element.

  ⌋

- `StructuredReq`: This represents a structured requirement as it is used within AUTOSAR RS documents.

Note that as `TraceableText` is aggregated in `DocumentationBlock` it also requires a proper rendition in printed documents. For an example of a proper rendition see [TPS_STDT_0001] above.

[constr_2565] Trace shall not be nested ⌈ Due to the intended atomicity of requirements respectively specification items, `Trace` shall not be nested. ⌋

[TPS_STDT_0042] namePattern for `shortNames` of `TraceableText` in Template Documents ⌈ The intended name pattern applicable to short names `TraceableText` (in fact representing e.g. requirement tags) in AUTOSAR standardization documents is defined as

```
{keyword(TraceCategory)}_{module}_{index}
```

In this pattern, the placeholders are defined as:

- `keyword(TraceCategory)` is defined in [4] in keyword set `InformationCategories`, entries with classification `TraceCategory`.
- `module` is either module abbreviation in [5] or an entry of the keyword set `DocumentAbbreviations` with classification `DocumentAbbreviation` in [4].
- `index` is a four digit numerical index

Note that this pattern is not yet applied in all AUTOSAR Documents. ⌋*(RS_STDT_0009, RS_STDT_0008, RS_STDT_0001)*

[TPS_STDT_0052] Characteristics of TraceableText ⌈ TraceableText should[1] be:

- **identifiable**: `TraceableText` shall be identified by a unique short name (see [TPS_STDT_0042]). This is automatically fulfilled by applying the AUTOSAR meta model and schema.
- **specific**: `TraceableText` should be written such that the content is unambiguous and comprehensive - even if this would not result in an elegant writing style.
- **atomic**: One `TraceableText` should cover one particular issue.
- **verifiable**: The content of `TraceableText` should be written concrete such that it can be verified - not necessarily automatically but at least by human experts.

  In particular the requirement levels specified in [TPS_STDT_0053] shall be applied.

⌋*(RS_STDT_0008, RS_STDT_0009)*

[TPS_STDT_0053] Expression of obligation ⌈The following verbal forms for the expression of obligation shall be used to indicate requirements.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as follows, based on [3].

Note that the requirement level of the document in which they are used modifies the force of these words.

- MUST: This word, or the adjective "LEGALLY REQUIRED", means that the definition is an absolute requirement of the specification due to legal issues.
- MUST NOT: This phrase, or the phrase "MUST NOT", means that the definition is an absolute prohibition of the specification due to legal issues.
- SHALL: This phrase, or the adjective "REQUIRED", means that the definition is an absolute requirement of the specification.
- SHALL NOT: This phrase means that the definition is an absolute prohibition of the specification.
- SHOULD: This word, or the adjective "RECOMMENDED", means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED", means that there may exist valid reasons in particular circumstances when the particular be-

[1] This usage of the word "should" indicates that this is not always easy to decide. For example [TPS_STDT_0052] could also have been divided in one `TraceableText` per item.

havior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

- MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item.

An implementation, which does not include a particular option, SHALL be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, SHALL be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

⌋

[TPS_STDT_0054] Organisation of `TraceableText` ⌈ A set of `TraceableText` within a specification shall have the following properties:

- **hierarchical structure:** Multiple `TraceableText`s shall be structured in several successive levels - this is mostly ensured by the templates for the different kind of AUTOSAR specifications.
- **completeness:** `TraceableText` at one level shall fully implement all `TraceableText` of the previous level.
- **external consistency:** Multiple `TraceableText`s shall not contradict each other.
- **no duplication of information within any level of the hierarchical structure**: The content of one `TraceableText` shall not be repeated in any other `TraceableText` within the same level of the hierarchical structure.
- **maintainability:** A set of `TraceableText` can be modified or extended, e.g. by introduction of new versions of TraceableText or by adding/removing `TraceableText`. The `shortName` of `TraceableText` shall not be reused or changed.

⌋

The levels mentioned in [TPS_STDT_0054] are illustrated in figure 2.1.

[TPS_STDT_0050] namePattern for AUTOSAR delivered Files ⌈ The intended name pattern applied for filenames of AUTOSAR delivered files is defined as

```
AUTOSAR_{keyword(DocumentCategory)}_{DocumentName}
```

In this pattern, the placeholders are defined as:

- `keyword(DocumentCategory)` is defined in [4] in keyword set `InformationCategories`, entries with classification `DocumentCategory`.

- `DocumentName` is the `shortName` of the `Keyword` according to [4], keyword set `DocumentAbbreviation` entries with classification `DocumentAbbreviation` or the shortName of the module in [5]

⌋(*RS_STDT_0009*)

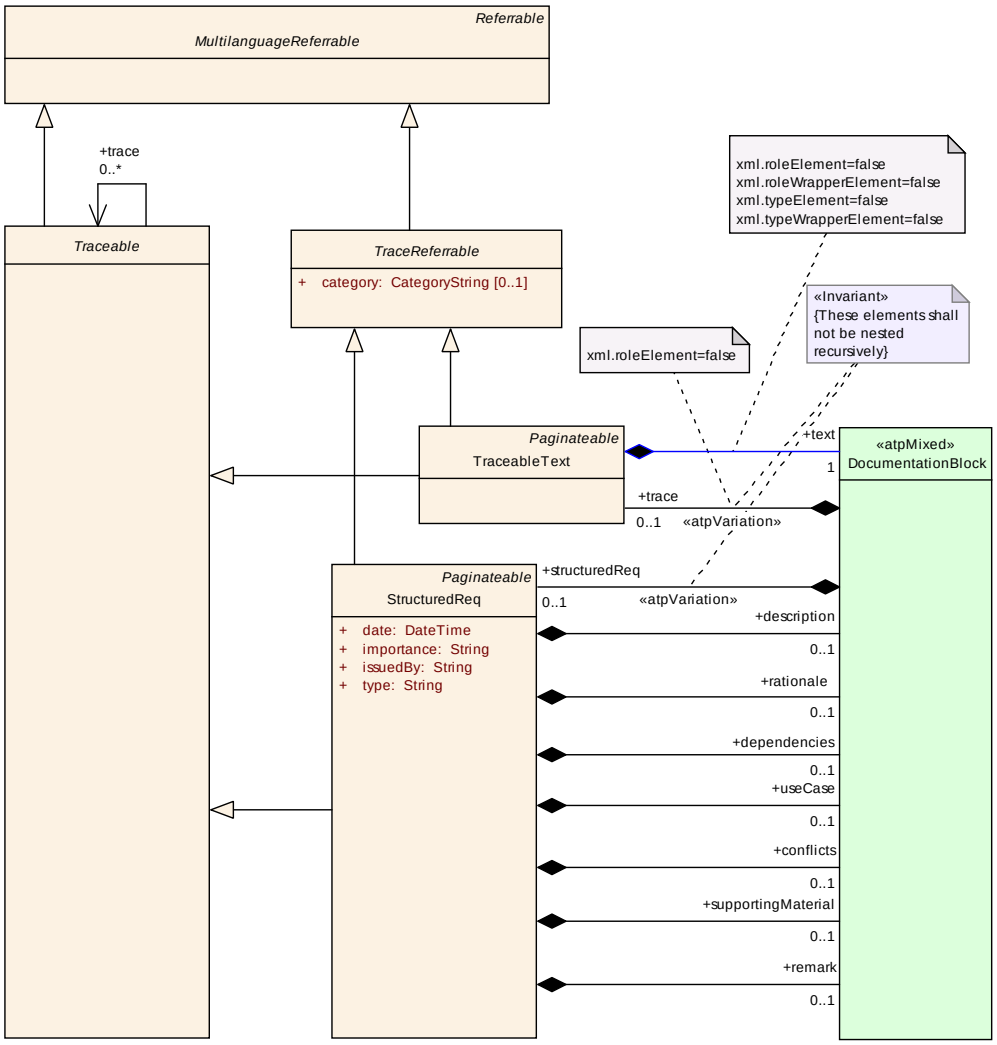

Figure 2.2: Requirements and Tracing

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***

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

Table 2.1: 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,MultilanguageReferrable,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.roleWrapperElement=false; xml.sequenceOffset=30; xml.typeElement=false; xml.typeWrapperElement=false

Table 2.2: 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,MultilanguageReferrable,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

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
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
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 2.3: StructuredReq

3 The Principles of Blueprints

[TPS_STDT_0002] The Principles of Blueprints ⌈ This chapter describes the support of the AUTOSAR meta-model for the pre-definition of model elements taken as the basis for further modeling. These pre-defintions are called blueprints. ⌋*(RS_STDT_0001)*

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

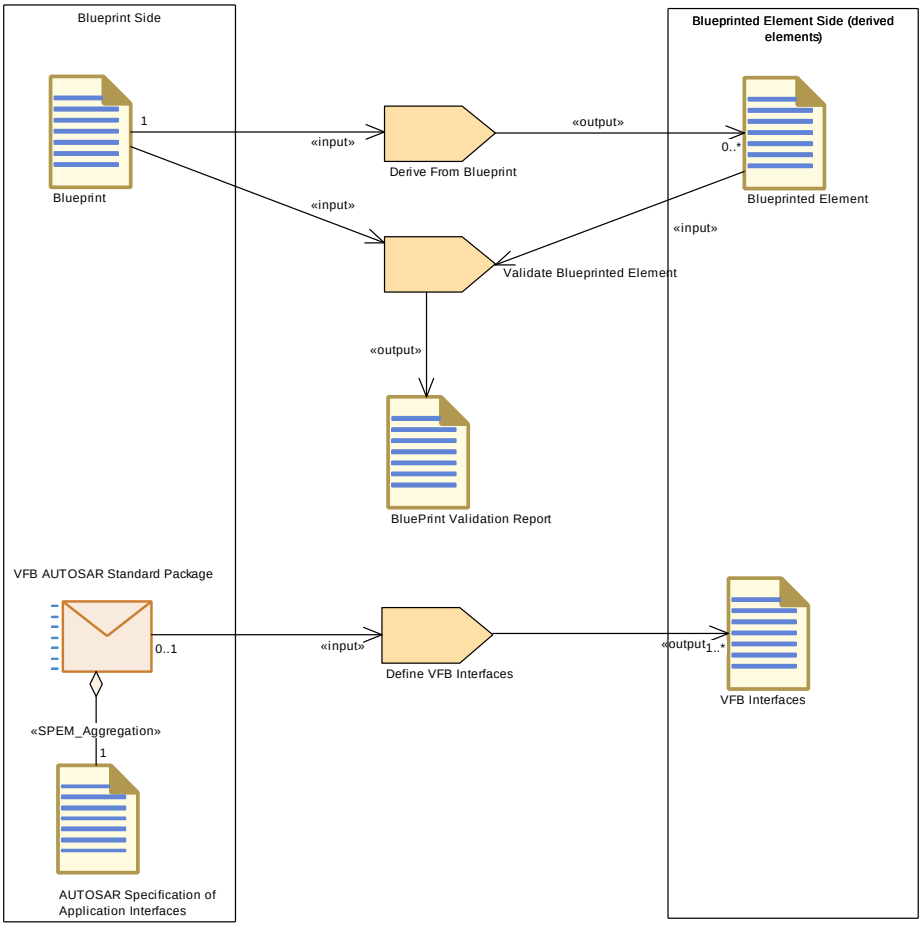

Figure 3.1: Blueprint methdology approach

Figure 3.1 illustrates the usecase. The blueprint is on one hand used as an input to derive objects (DeriveFromBlueprint) and later also used to validate the derived objects. As an Example the figure shows that the Application interfaces are used to derive VFB interfaces (namely `PortInterfaces`).

3.1 Abstract pattern for Blueprints

The blueprint approach is represented by the abstract blueprint structure as shown in figure 3.2. It is based on three entities:

- **Blueprint**, represented by `AtpBlueprint`, acts as the predefinition of the element. Basically it follows the same structure as the derived elements.

  But there might be additional elements to support the fact that it is a blueprint. An example for this is that `PortPrototypeBlueprint` also specifies `initValue`s which is not the case for `PortPrototypes` which get their init values from appropriate `ComSpec`s.

- **Blueprinted Element**, represented by `AtpBlueprintable`, acts as the element which was derived from the Blueprint. These elements are derived from

blueprints mainly by copy and refine. This "refine" may add further attribute values, update `shortName` etc. The details of possible refinements are specified for each blueprint individually.

Note that the subsequent processing of blueprinted elements (e.g. RTE generation) do not refer to the blueprints anymore.

- **Blueprint Mapping**, represented by `AtpBlueprintMapping`, acts as a reference between blueprints and their derived elements. The main purpose of this blueprint mapping is to
  - provide the ability to validate for each derived element that they conform to the blueprint.
  - reflect the fact that the derived elements are part of a common concept.

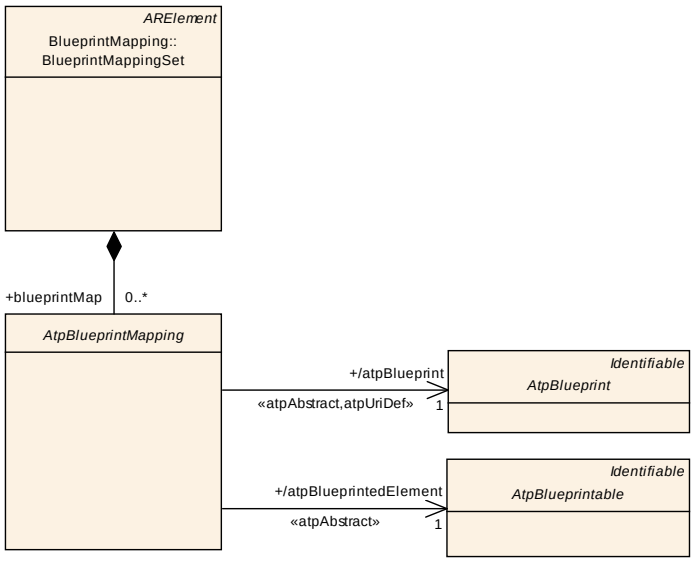

Figure 3.2: Abstract Blueprint Structure

Meta-classes for elements eligible for blueprinting are defined as specializations of `AtpBlueprintable` while meta-classes for blueprints are defined as specializations of `AtpBlueprint`. An example is given in figure 3.3.

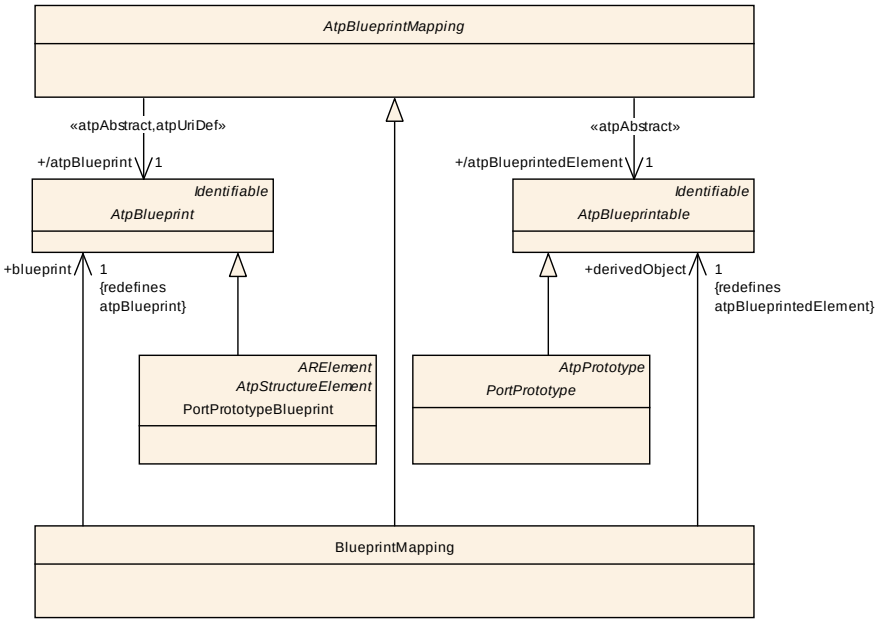

Figure 3.3: Port Blueprints as an example for separate meta-classes for Blueprint and blueprinted Element

For most of the elements eligible for blueprinting, no extra meta-class is required because the same meta-class applies for blueprints and blueprinted elements. The meta-class of such an element inherits from both `AtpBlueprint` and `AtpBlueprintable`. An example is given in figure 3.4.

[TPS_STDT_0041] Constraints may be violated in Blueprints ⌈ For blueprints using the same meta-class as the derived objects, the constraints defined for these objects may be violated by the blueprints such as:

- Required attributes may be missing (For this reason, such blueprints also may violate the strict AUTOSAR schema).
- Referenced objects may not exist. Strictly speaking, references in blueprints can all be considered as `atpUriDef`

⌋*(RS_STDT_0002, RS_STDT_0006, RS_STDT_0007)*

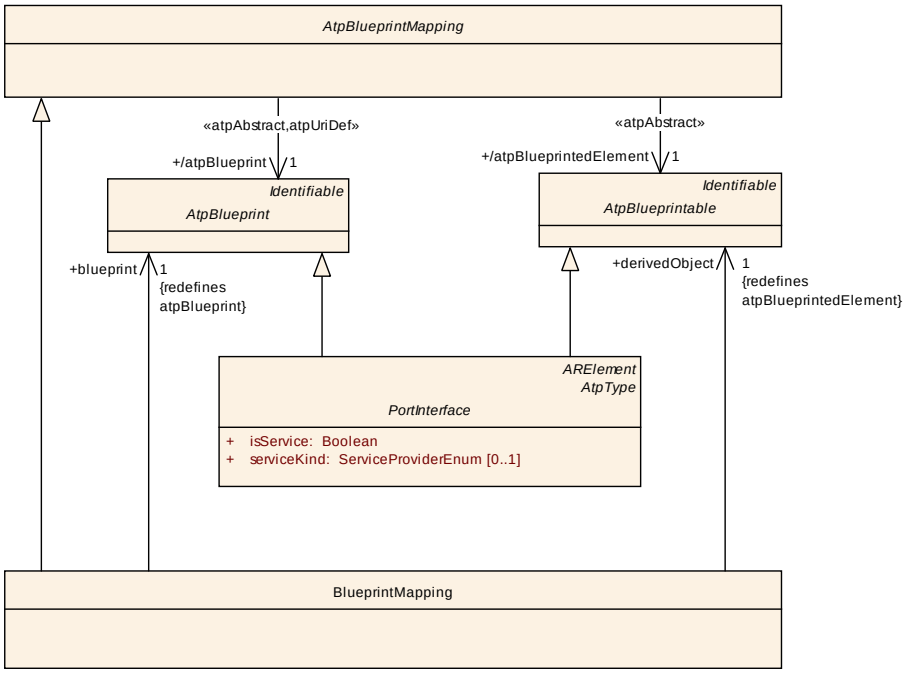

Figure 3.4: PortInterface Blueprints as an example for using the same meta-class for Blueprint and blueprinted Element

[TPS_STDT_0033] Recognize Blueprints ⌈ According to [6] the blueprints reside in a package of `category` "BLUEPRINT". Downstream AUTOSAR Tools such as RTE-generator shall ignore Elements living in a package of `category` "BLUEPRINT". ⌋*(RS_STDT_0006, RS_STDT_0007)*

Blueprints are specializations of `AtpBlueprint`. Introduction of standardization therefore does not introduce compatibility problems to existing templates. Note that since AUTOSAR 4.0.3 `AtpBlueprint.shortNamePattern` is replaced by `Identifier.namePattern` resp. `CIdentifier.namePattern`.

[TPS_STDT_0047] Ignore Blueprint Attributes ⌈ AUTOSAR Tools which do not process blueprints such as RTE-generator shall ignore `Identifier.namePattern` resp. `CIdentifier.namePattern`.

The attributes `Identifier.namePattern` resp. `CIdentifier.namePattern` should be removed when deriving objects from blueprints. ⌋*(RS_STDT_0003, RS_STDT_0004, RS_STDT_0006, RS_STDT_0007)*

Class	**AtpBlueprint (abstract)**			
Package	M2::AUTOSARTemplates::StandardizationTemplate::AbstractBlueprintStructure			
Note	This meta-class represents the ability to act as a Blueprint. As this class is an abstract one, particular blueprint meta-classes inherit from this one.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	Datatype	Mul.	Kind	Note

shortName Pattern	String	0..1	attr	This attribute represents the pattern which shall be used to build the shortName of the derived elements. As of now it is modeled as a String. In general it should follow the pattern:
				`pattern = (placeholder | namePart)*`
				`placeholder = "{" namePart "}"`
				`namePart = identifier | "_"`
				
				This is subject to be refined in subsequent versions.
				
				Note that this is marked as obsolete. Use the xml attribute namePattern instead as it applies to Identifer and CIdentifer (shortName, symbol etc.)
				
				Tags: atp.Status=obsolete

Table 3.1: AtpBlueprint

Class	AtpBlueprintable (abstract)			
Package	M2::AUTOSARTemplates::StandardizationTemplate::AbstractBlueprintStructure			
Note	This meta-class represents the ability to be derived from a Blueprint. As this class is an abstract one, particular blueprintable meta-classes inherit from this one.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 3.2: AtpBlueprintable

Class	AtpBlueprintMapping (abstract)			
Package	M2::AUTOSARTemplates::StandardizationTemplate::AbstractBlueprintStructure			
Note	This meta-class represents the ability to express a particular mapping between a blueprint and an element derived from this blueprint. Particular mappings are defined by specializations of this meta-class.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
atpBlueprint	AtpBlueprint	1	ref	This represents the blueprint. **Stereotypes:** atpAbstract **Tags:** xml.sequenceOffset=50
atpBlueprintedElement	AtpBlueprintable	1	ref	This represents the bluprinted elements which shall be mapped to the blueprint. **Stereotypes:** atpAbstract **Tags:** xml.sequenceOffset=60

Table 3.3: AtpBlueprintMapping

3.2 Mapping of Blueprints to blueprinted Elements

In many cases it will be necessary to identify the relationship of a blueprinted element (e.g. `PortPrototype`) to the corresponding blueprint (e.g. `PortPrototypeBlueprint`) after the blueprinted element has been created according to the blueprint.

For this purpose it would theoretically be possible to establish a reference from `AtpBlueprintable` to `AtpBlueprint` that identifies the pair of related model artifacts. However, this kind of information is relevant only in a narrow scope and does - as mentioned before - not impact the downstream model handling.

Therefore, a `AtpBlueprintMapping` is introduced which refers to both `AtpBlueprintable` and `AtpBlueprint` (see figure 3.2). The `AtpBlueprintMap` is in turn aggregated at a container for the creation of blueprint mappings, the `BlueprintMappingSet`.

In previous AUTOSAR Releases a specialization of `AtpBlueprintMap` was created for each particular meta class eligible for blueprinting. This has been replaced by one particular specialization (`BlueprintMapping`)[1].

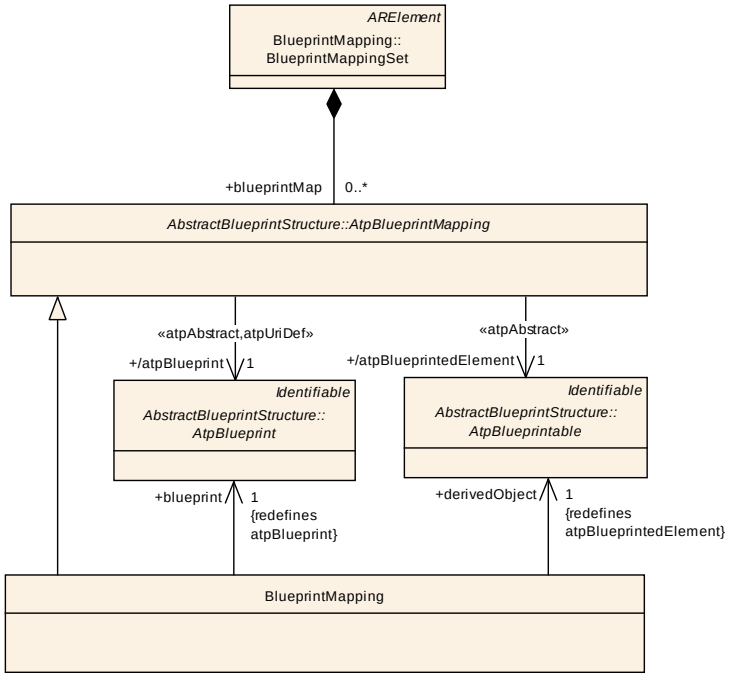

Figure 3.5: Mapping of Derived Objects and their Blueprints

[constr_2566] Blueprintmapping shall map appropriate elements ⌈ `BlueprintMapping` shall map elements which represent a valid pair of blueprint / derived object. In most of the cases this means that `blueprint` and `derivedObject` shall refer to objects of the same meta-class. ⌋

[1] For compatibility reasons, the abstract patten was not changed. The previous specializations (`PortInterfaceBlueprintMapping` and `PortPrototypeBlueprintMapping` are obsolete, but kept in the schema.

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 3.4: BlueprintMappingSet

3.3 General Rules for Compliance of blueprint and blueprinted element

[TPS_STDT_0005] Compliance with Blueprints ⌈ Constraints [constr_2554] and [constr_2555] apply in general for the compliance of blueprints with the derived objects. ⌋*(RS_STDT_0017)*

[constr_2554] Derived objects shall match the blueprints ⌈ Unless specified explicitly otherwise, the attributes the blueprint shall appear in the derived objects.

As an exception `namePattern` may **not** be copied. ⌋

[constr_2555] Derived objects may have more attributes than the blueprints ⌈ Unless specified explicitly otherwise, derived objects may have more attributes than the blueprints. Such attributes can be

- additional values if the upper multiplicity of the attribute in the meta-model is greater than 1
- those specified by the related templates but not specified in the blueprint

⌋

[constr_2542] Compatibility of `introduction` of blueprint and blueprinted element ⌈ Elements derived from blueprints are allowed to extend the `introduction` such that additional content is added at the bottom only. ⌋

Note that `Introduction` should not be used to describe the derivation of objects from the blueprint. This is done in `blueprintCondition` resp. `blueprintValue`. See [TPS_STDT_0048] for details.

[constr_2543] Specify a name pattern in blueprints ⌈ For each blueprint, a `namePattern` shall be specified if the `shortName` respectively a `symbol` is not fixed but intended to be defined when objects are derived from a blueprint. This is used to verify the appropriate naming of the derived objects ([constr_2553]). ⌋

[constr_2553] `shortName` shall follow the pattern defined in the Blueprint ⌈ The `shortName` respectively `symbol` of the derived objects shall follow the pattern defined in `namePattern` of the blueprint according to [constr_2543] ⌋

[constr_2570] No Blueprints in system descriptions ⌈ There shall be no blueprints in system descriptions. In consequence of this blueprint elements shall be referenced only from blueprints and `BlueprintMap`s. Due to ≪`atpUriDef`≫, the references from `BlueprintMap` do not need to be resolved in system descriptions. ⌋

[constr_2571] Outgoing references from Blueprints ⌈ Note that outgoing references from Blueprints are basically not limited. Practically, references to objects living in a package of category EXAMPLE should not occur. ⌋

Reason for [constr_2571] is the fact that these examples then also shall exist in the target system description but not as example. In such a case the example would take the role of a blueprint.

Figure 3.6 illustrates a scenario with standardized objects, blueprints and project related objects.

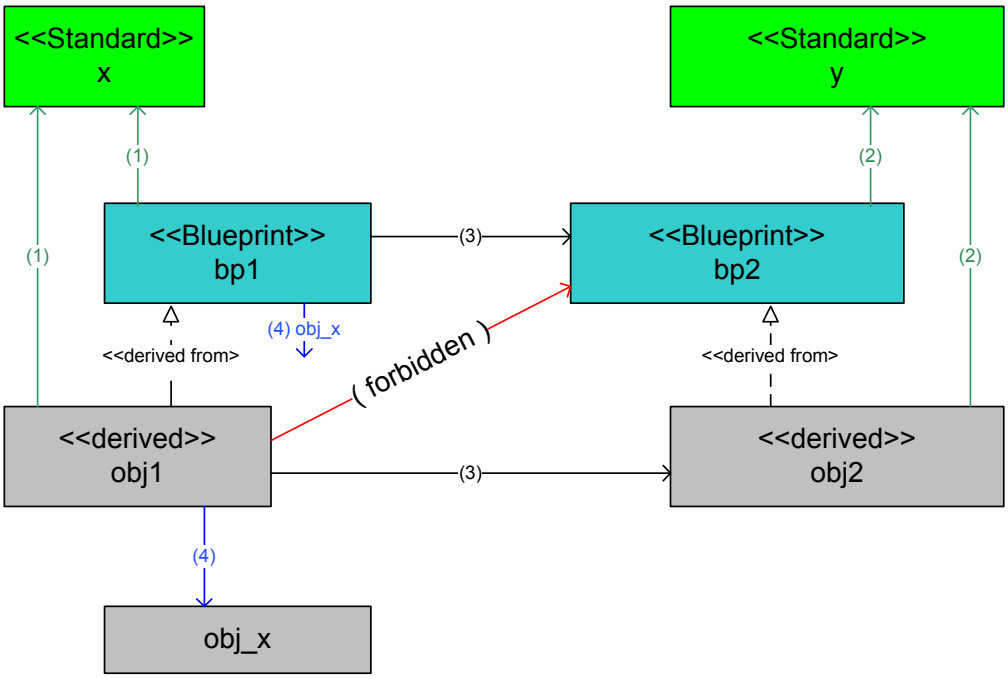

Figure 3.6: Relations between Blueprints, "Derived Objects" and "Standardized Objects"

This diagram in particular illustrates how references in blueprints shall be handled:

[TPS_STDT_0051] Handling references when deriving objects from blueprints ⌈

- Blueprints may reference standardized objects. These references also exist in the derived objects (1), (2).
- Blueprints may reference other blueprints (3). These references need to be replaced in order to meet [constr_2546]. Therefore a reference from a derived object to a blueprint is not allowed.

- Blueprints may contain references to arbitrary objects (4). According to [TPS_STDT_0041] it is allowed that these objects even do not exist. Nevertheless to meet [constr_2554] such references shall be copied to the derived objects and the referenced objects shall exist in the target system description.

⌋*(RS_STDT_0013, RS_STDT_0017)*

[TPS_STDT_0034] Integrity of Blueprints ⌈ The integrity of blueprints can be established by applying references to blueprints of related objects. For example, a blueprint of a `BswModuleDescription` may refer to a blueprint of `BswModuleEntry`. ⌋*(RS_STDT_0027)*

[constr_2546] References from Blueprint to Blueprint need to be replaced in derived objects ⌈ A blueprint may refer to another blueprint. When deriving objects such a reference shall be replaced such that the new reference target is an object derived from the corresponding reference target in the blueprint. ⌋

[TPS_STDT_0048] Express Decisions when Deriving Objects ⌈ Applying `VariationPoint` is a suitable way to express intended decisions to be made when deriving objects from blueprints. In this case the value of the UML tag `Vh.latestBindingTime` is `BlueprintDerivationTime` and `VariationPoint.blueprintCondition` respectively `AttributeValueVariationPoint.blueprintValue` shall be used to express the intended derivation. ⌋*(RS_STDT_0008, RS_STDT_0018, RS_STDT_0019)*

[TPS_STDT_0028] Resolving `VariationPoint` in Blueprints ⌈ If a `VariationPoint` has only `blueprintValue` respectively `blueprintCondition` but not `swSyscond` nor `postBuildVariantCondition` it shall be resolved when deriving elements. ⌋*(RS_STDT_0014, RS_STDT_0015, RS_STDT_0019, RS_STDT_0020)*

Please refer to Generic Structure Template [6] for the following aspects:

- Even if `BindingTimeEnum` does not contain the value `BlueprintDerivationTime`, there are sill `VariationPoint`s which shall be bound on blueprint derivation. This is specified as `BlueprintDerivationTime` in the UML tag `Vh.latestBindingTime` at the variation point in the meta model.
- In [constr_2537] `VariationPoint` is limited to `SwComponentType`, `BSWmoduleDescription`, `Documentation`, even if the meta model supports variation point on any `PackageableElement`.

  [constr_2564] `VariationPoint` in Blueprints of `PackageableElements` ⌈ To support standardization, constraint [constr_2537] in [6] is relaxed for blueprints. This means in particular, that all `PackageableElements` which inherit from `AtpBlueprint` and live in a package of category BLUEPRINT may have a `VariationPoint`.

  In this case `Vh.latestBindingTime` is considered `BlueprintDerivationTime` even if the meta model still states `SystemDesignTime` for `PackageableElements`. ⌋

See chapter 4 for such elements.

- [constr_2557]: System configurations shall not contain `VariationPoint`s with `Vh.latestBindingTime` set to `BlueprintDerivationTime`.
- [constr_2558]: If `Vh.latestBindingTime` is `BlueprintDerivationTime` then there shall only be `blueprintCondition/blueprintValue`.
- [constr_2559]: `VariationPoint` shall not be nested. In particular this means that there shall not exist a `VariationPoint` within the `DocumentationBlock` in the role `blueprintCondition` in a `VariationPoint`.
- [constr_2567]: Attribute Value Blueprints should contain `undefined`.

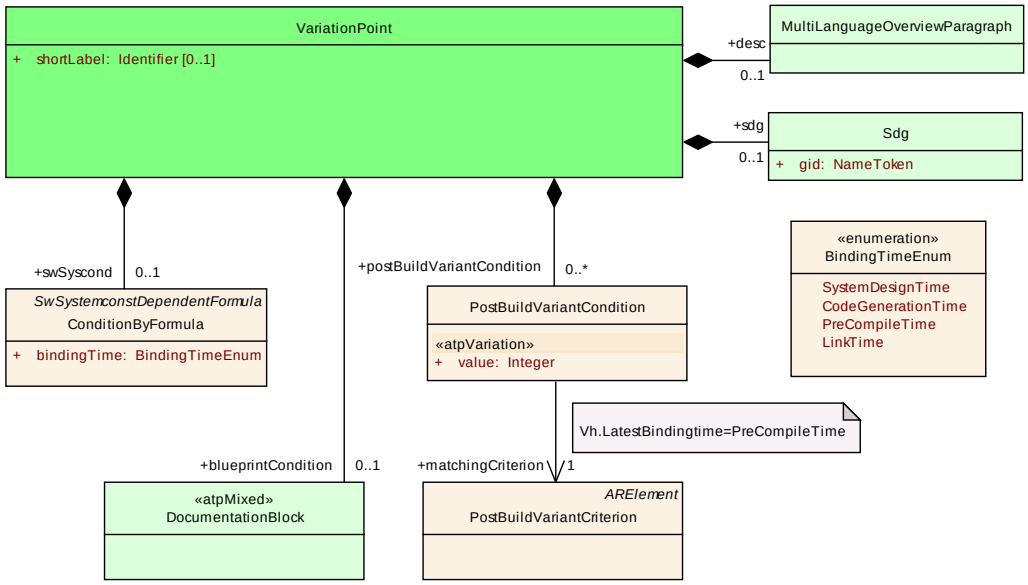

Figure 3.7: Variation Point

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
blueprintCondition	Documentation Block	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

Attribute	Datatype	Mul.	Kind	Note
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 3.5: VariationPoint

[TPS_STDT_0030] Blueprint of `VariationPoint` ⌈ A blueprint may contain `VariationPoint` with `Vh.latestBindingTime` set to `BlueprintDerivationTime`. These are considered as kind of blueprint of variation points which shall be handled when deriving objects. The following options apply for the container of the `VariationPoint` according to the information provided in `VariationPoint.blueprintCondition`:

- is resolved manually when deriving objects.
- is resolved by a module generator. The resolver approach is not formalized but hard coded in the module generator. Note that in this case it is also likely that multiple objects are created by the module generator. This shall also be noted in the `blueprintCondition`.
- is converted to a subsequent VariationPoint

⌋*(RS_STDT_0020)*

[TPS_STDT_0044] Transferring `VariationPoint` ⌈ Unless specified explicitly otherwise, `VariationPoint`s with `Vh.latestBindingTime` **not** set to `BlueprintDerivationTime` should be transferred to the derived objects (see also [con-

str_2555]). Thereby the `shortLabel` of the `VariationPoint` may be adapted according to the description in the `blueprintCondition`. ⌋*(RS_STDT_0020)*

[constr_2556] No Blueprint Motivated `VariationPoints` in AUTOSAR Descriptions ⌈ AUTOSAR descriptions which are not blueprints shall not have `blueprintCondition` nor `blueprintValue`. ⌋

[constr_2569] Purely Bluprint Motivated `VariationPoints` ⌈ `VariationPoints` with `Vh.latestBindingTime` set to `BlueprintDerivationTime` shall have only `blueprintCondition` respectively `blueprintValue`. ⌋

[TPS_STDT_0045] Transferring Objects in General ⌈ Objects resp. references without `VariationPoint` shall be transferred to the derived objects. Thereby the `namePatterns` of the referenced Blueprints also apply for rewriting the shortName path in the reference. ⌋*(RS_STDT_0020)*

For more details about `VariationPoint` refer to [6], as all constraints are summarized there.

[TPS_STDT_0046] Configuration dependent properties ⌈ Some data types specify configuration-dependent properties like limits, base types etc.

This is supported by an additional attribute `blueprintValue` in the `AttributeValueVariationPoint`. This attribute correlates to `blueprintCondition` in `VariationPoint`. ⌋*(RS_STDT_0020)*

An example for [TPS_STDT_0046] is:

```
NvM_BlockIdType Range:  0..2\^(16- NvMDatasetSelectionBits)-1
Dem_RatioIdType Type: uint8, uint16
```

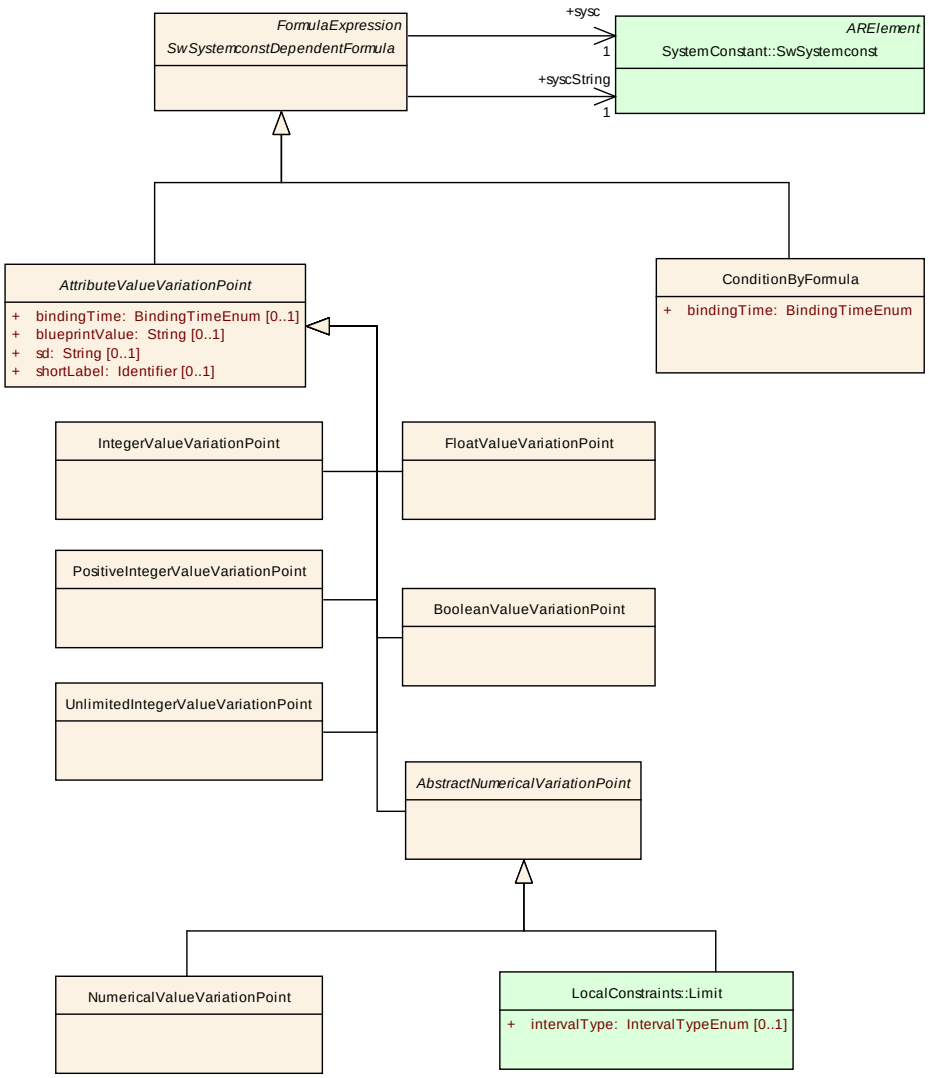

Figure 3.8: Attribute Value Variation Point

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

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
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 3.6: AttributeValueVariationPoint

3.4 Applicable patterns to define names when deriving objects from blueprints

[TPS_STDT_0003] Applying `namePattern` ⌈ When deriving an element from a blueprint it is often the case that a particular pattern shall be used to determine the `shortName` respectively the `symbol` of the object. This use case is supported by the attribute `namePattern` in `Identfier` resp. `CIdentifier`. ⌋*(RS_STDT_0004, RS_STDT_0008, RS_STDT_0019, RS_STDT_0021)*

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***

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
namePatte rn	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 3.7: Identifier

Primitive	CIdentifier			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Primitive Types			
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*
namePatte rn	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 3.8: CIdentifier

[TPS_STDT_0055] General Syntax for Name Patterns ⌈ The name pattern uses the following syntax defined according to ANTLR [7].

Listing 3.1: Grammar for name pattern

```
grammar NamePattern;


namePattern
        : (fixedName | placeholder | separator)+ ;

subPattern
  : '('  (fixedName | placeholder | separator )+  ')' ('?' | '*' | '+')? ;

placeholder  : '{'
                 ('anyName' |
                  'blueprintName' |
                  'componentName' |
                  'ecucValue' '(' ecucName ')' |
                  'index' |
                  'keyword' '(' kwClass ')' |
                  'modeName' |
                  'nameSpace' |
```

```
                'portDir' |
                'typeId'   |
                subPattern
              )
            '}' ;

fixedName :  MyName;

kwClass :    MyName;

separator
  :    Separator ;

ecucName: EcucName;

EcucName :     (('a'..'z') | ('A'..'Z'))  ( 'a'..'z' | ('A'..'Z') | ('0'..'
    9') | '_'  | '/' )*;

MyName  :  (('a'..'z') | ('A'..'Z'))  ( 'a'..'z' | ('A'..'Z') | ('0'..'9')
    | '-' )*;


Separator
  : '_' ;
```

⌋*(RS_STDT_0004)*

Example 3.1 illustrates valid name patterns. Note that `{blueprintName}` etc. denotes a placeholder.

Example 3.1

```
{blueprintName}_{anyName}

{portDir}_{blueprintName}_{keyword(Qualifier)}_{componentName}_{index}
  --> example for a match:  R_EngN_Max_Dem_3

{componentName}_{ecucValue(item1)}

h_b_{(a_{index}_b_{componentName}_{(x_{ecucValue(hugo)})*})*}
```

The semantics of the placeholder is defined as follows:

anyName This represents a string which is valid `shortName` according to `Identifier`

blueprintName This represents the `shortName` / `shortLabel` / `symbol` of the applied blueprint

componentName This represents the `shortName` of the BSW module resp. ASW SwComponentType / ASW component prototype related to the derived object. "Related" mainly could be both, aggregating or referencing.

[TPS_STDT_0036] Placeholder for Module / Component ⌈ The placeholder `componentName` in particular supports multiple derivation of a `PortPrototypeBlueprint` in the context of different software component types resp. modules. ⌋*(RS_STDT_0021)*

ecucValue [TPS_STDT_0040] Influence of ECUC ⌈ This indicates an influence of the ECU configuration. This placeholder takes an argument which is intended as a keyword reflecting the kind of influence. More details shall be specified in the `blueprintCondition` where the argument mentioned before can be taken for reference. ⌋*(RS_STDT_0002, RS_STDT_0010)*

index This represents a numerical index applicable for example to arrays

keyword [TPS_STDT_0004] Abbreviated Name ⌈ This represents the `abbrName` of a keyword acting as a name part of the short name. The eligible keywords can be classified (using the argument `kwClass`). This classification shall match with one of the `classification` of the applied keyword. ⌋*(RS_STDT_0005)*

modeName This represents the shortName of the mode e.g. `Dcm_{modeName}ModeEntry`

portDir This represents the direction of a port.

[TPS_STDT_0037] Port Direction ⌈ The placeholder `portDir` in particular supports the case that the same blueprint is used for P-Port as well as for an R-Port. The values represented by this placeholder is `P` for P-Port respectively `R` for R-Port. ⌋*(RS_STDT_0021)*

typeId This represents an indicator based on the type of the object

3.5 Ecu Configuration Parameters and Blueprints

[TPS_STDT_0025] Deriving VSMD from STMD Uses its own Mechanism ⌈ Basically the Standard Module Definitions (STMD) specified by AUTOSAR according to [8] could also be considered as blueprints. On the other hand, the relationship between vendor specific module defintions (VSMD) is a very strict one and was there before the general concept of Blueprints was introduced. Therefore for sake of compatibility this relationship is still maintained using `EcucModuleDef.refinedModuleDef`.

Nevertheless for company specific applications there is some support for ECU configuration in Standardization Template. ⌋*(RS_STDT_0022, RS_STDT_0010)*

See chapter 4.9 resp. chapter 4.10 for more details.

4 Blueprints defined in AUTOSAR

The following sub chapters specify the particular model elements for which blueprints are supported.

4.1 Blueprinting AliasNameSet

[TPS_STDT_0011] Blueprinting `AliasNameSet` ⌈ `AliasNameSet` can be blueprinted. ⌋*(RS_STDT_0023)*

4.2 Blueprinting ApplicationDataType

[TPS_STDT_0023] Blueprinting `ApplicationDataType` ⌈ `ApplicationDataType` can be blueprinted. ⌋*(RS_STDT_0028, RS_STDT_0029)*

4.3 Blueprinting ARPackage

[TPS_STDT_0013] Blueprinting `ARPackage` ⌈ `ARPackage` can be blueprinted. Main use case is to support predefined package structures, e.g. those specified in [6]. ⌋*(RS_STDT_0013, RS_STDT_0030)*

4.4 Blueprinting BswModuleDescription

[TPS_STDT_0027] Blueprinting `BswModuleDescription` ⌈ `BswModuleDescription` can be blueprinted. ⌋*(RS_STDT_0001)*

Blueprints for `BswModuleDescription` are used in particular to describe dependencies to to other modules. Note that in this case all references to other modules and module entries are targeting blueprints of the intended module. These references need to be replaced when deriving objects from `BswModuleDescriptionBlueprint`.

A blueprint of `BswModuleDescription` shall specify the references to the standard- or blueprint- API elements, in particular

- `BswModuleDescription.providedEntry`
- `BswModuleDescription.outgoingCallback`
- `BswModuleDescription.bswModuleDependency.requiredEntry`
- `BswModuleDescription.bswModuleDependency.expectedCallback`

Nevertheless, it is allowed that derived `BswModuleDescription` adds further ones of these references.

Furthermore, optional elements like callbacks often come in 0..* multiplicity. In this case, the blueprint should specify one callback reference (to one blueprint BswModuleEntry) and express the open multiplicity in its `namePattern` respectively in the `VariationPoint.blueprintCondition` as illustrated in Figure 4.1.

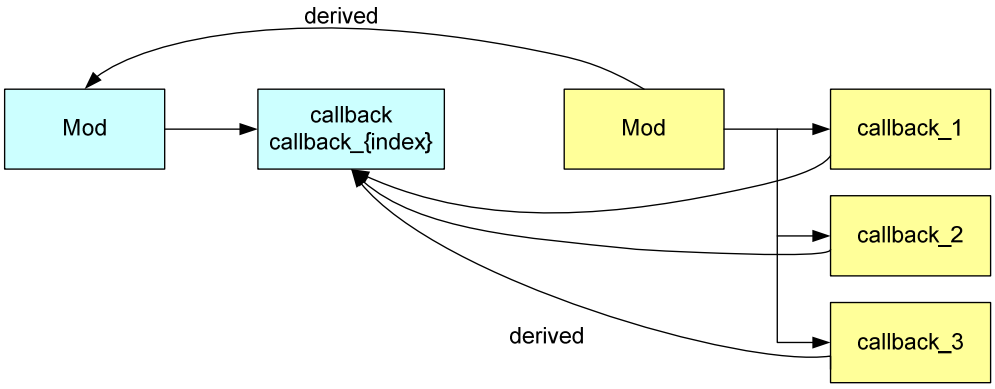

Figure 4.1: Multiply derived Objects

[constr_2563] `BswModuleDescription` blueprints should not have a `BswModuleBehavior` ⌈ A `BswModuleDescription` blueprint should not have a `BswModuleBehavior` since this is a matter of implementation and not subject to standardization. Exceptions might exist in vendor internal applications. ⌋

4.5 Blueprinting BswModuleEntry

[TPS_STDT_0014] Blueprinting `BswModuleEntry` ⌈ `BswModuleEntry` can be blueprinted. ⌋*(RS_STDT_0002, RS_STDT_0018, RS_STDT_0029)*

The meta-class `BswModuleEntry` and its composites (`SwServiceArg`) contain optional as well as mandatory elements which are never or only sometimes standardized, e.g. executionContext, swServiceImplPolicy, parts of SwServiceArg.swDataDefProps. Nevertheless Standardization Template does not explicitly specify constraint which attributes shall, may or shall not be defined in the blueprint (see also [TPS_STDT_0049]).

4.6 Blueprinting CompuMethod

[TPS_STDT_0015] Blueprinting `CompuMethod` ⌈ `CompuMethod` can be blueprinted. ⌋*(RS_STDT_0029)*

Sometimes it is required to extend a standardized enumeration with vendor specific elements.

For example [SWS_RamTst_192] states: If vendor specific algorithms were defined the enumeration fields of RamTst_AlgorithmType should be extended accordingly.

[TPS_STDT_0049] Blueprinting Enumerators ⌈ Extensions of enumerator values shall be expressed in the blueprint of the related `CompuMethod` by the `variationPoint` at `CompuScale` ⌋*(RS_STDT_0002, RS_STDT_0029)*

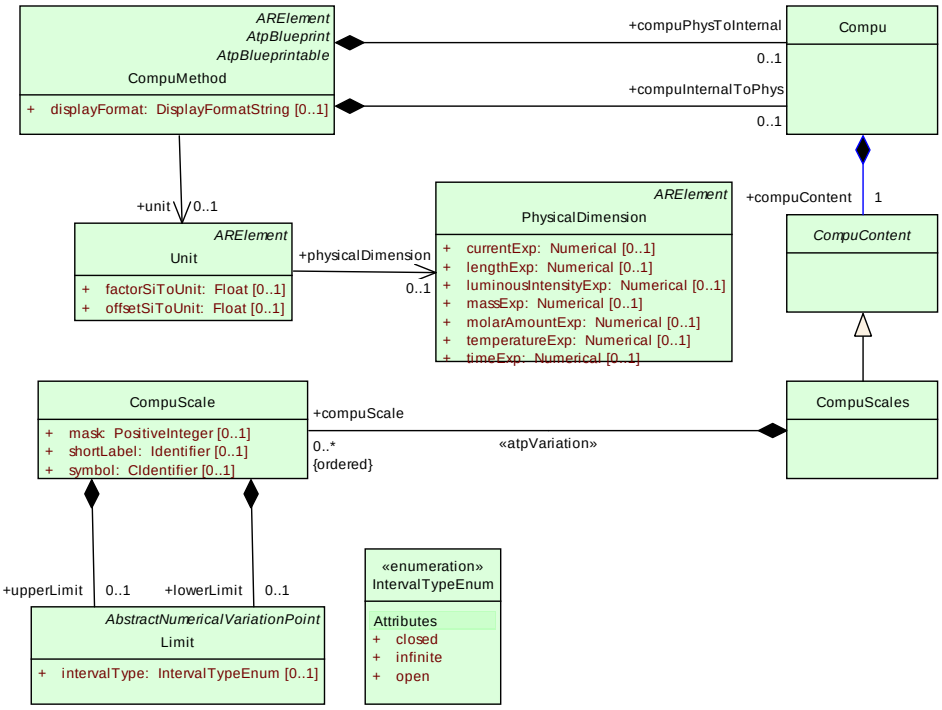

Figure 4.2: A `CompuMethod` and its attributes define data semantics

4.7 Blueprinting DataConstr

[TPS_STDT_0016] Blueprinting `DataConstr` ⌈ `DataConstr` can be blueprinted. ⌋*(RS_STDT_0029)*

4.8 Blueprinting DataTypeMappingSet

[TPS_STDT_0017] Blueprinting `DataTypeMappingSet` ⌈ `DataTypeMappingSet` can be blueprinted. ⌋*(RS_STDT_0029)*

4.9 Blueprinting EcucDefinitionCollection

[TPS_STDT_0018] Blueprinting `EcucDefinitionCollection` ⌈ `EcucDefinitionCollection` can be blueprinted. ⌋*(RS_STDT_0029)*

4.10 Blueprinting EcucModuleDef

[TPS_STDT_0019] Blueprinting `EcucModuleDef` ⌈ `EcucModuleDef` can be blueprinted. ⌋*(RS_STDT_0029)*

Note that this is intended for company internal use. Please refer to chapter 3.5.

4.11 Blueprinting FlatMap

[TPS_STDT_0035] Blueprinting `FlatMap` ⌈ `FlatMap` can be blueprinted. ⌋*(RS_STDT_0029)*

Usecase for blueprints of `FlatMap` is given in [9].

4.12 Blueprinting ImplementationDataType

[TPS_STDT_0020] Blueprinting `ImplementationDataType` ⌈ `ImplementationDataType` can be blueprinted. ⌋*(RS_STDT_0029)*

4.13 Blueprinting ModeDeclarationGroup

[TPS_STDT_0031] Blueprinting `ModeDeclarationGroup` ⌈ `ModeDeclarationGroup` can be blueprinted. ⌋*(RS_STDT_0024)*

4.14 Blueprinting PortPrototype

One of the major activities of the AUTOSAR initiative is the standardization of application interfaces. That is, in terms of the AUTOSAR meta-model the standardization mainly applies to the definition of `PortPrototype`s for specific purposes.

Due to the structure of the AUTOSAR meta-model it is not possible to merely express a standardized `PortPrototype` because for good reasons the latter does not exist on its own but is always owned by a `SwComponentType`.

Therefore, in the past the standardization of “application interfaces” involuntarily also involved the creation of `SwComponentType`s. This unnecessary complexity can be overcome by the usage of a `PortPrototypeBlueprint`.

[TPS_STDT_0007] Blueprinting `PortPrototype` ⌈ `PortPrototype` can be blueprinted by the specific meta class `PortPrototypeBlueprint`. ⌋*(RS_STDT_0003)*

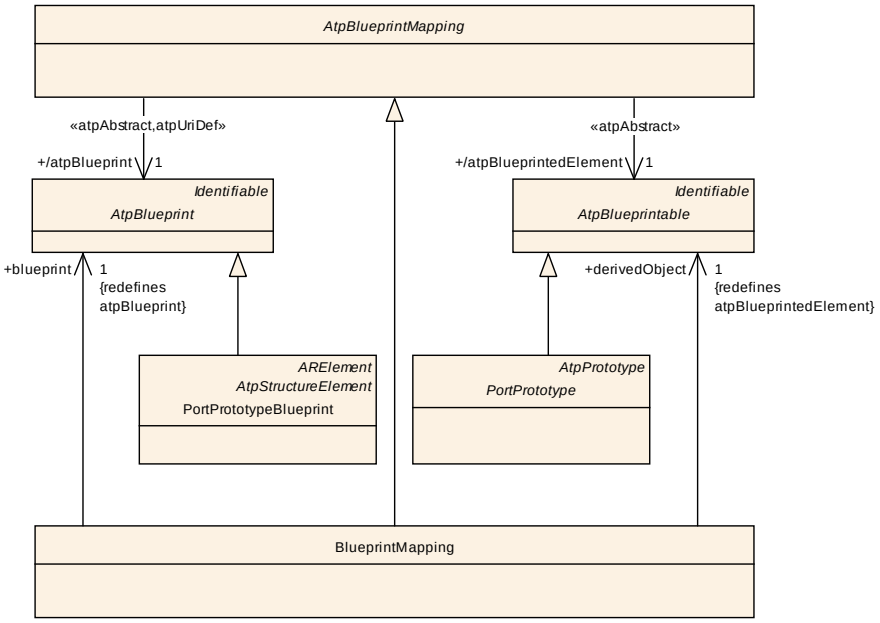

Figure 4.3: Mapping of Port Prototype Blueprints

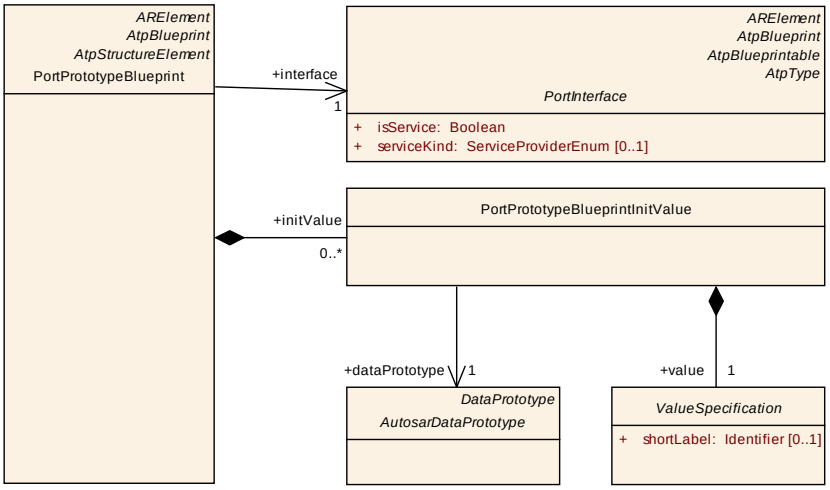

Figure 4.4: Blueprinting Port Prototype

A `PortPrototypeBlueprint` has the following characteristics:

- It is an `ARElement` and does therefore not require any element other than an `ARPackage` as context. It is therefore not necessary to involve “auxiliary” model elements into the definition of a standardized “application interface” for the mere purpose of conforming to the AUTOSAR meta-model.
- It acts as a “blueprint” for the creation of `PortPrototype`s. That is, probably supported by the used authoring tool, the user picks a specific `PortPrototypeBlueprint` and creates a `PortPrototype` out of it. The structure of the created `PortPrototype` is indistinguishable from a `PortPrototype` created without taking a `PortPrototypeBlueprint` as a blueprint. An `PortProto-`

typeBlueprint can be taken as the blueprint for as many PortPrototypes as required.

- It is possible to define additional attributes that are taken over to the created PortPrototype. For example, in some cases the definition of an initial value[1] is part of the definition of a standardized "application interface". Therefore, PortPrototypeBlueprint also supports the definition of an initValue, which needs to be moved to the appropriate ComSpecs.
- It has a reference to the corresponding PortInterface. If the referenced PortInterface is not a blueprint, it can directly be taken over by the PortPrototype created out of the PortPrototypeBlueprint such that the new PortPrototype references the PortInterface. If the referenced PortInterface is a blueprint, it is necessary to derive a PortInterface and reference this in the PortPrototype.
- It does not make any assumptions whether the PortPrototype created out of it will be a PPortPrototype or an RPortPrototype.
- It can basically be used for all kinds of PortInterfaces, i.e. it is not constrained to e.g. SenderReceiverInterfaces although this kind of PortInterface will most likely get a significant share of the usage of PortPrototypeBlueprint
- It can only be used for the standardization of “application interfaces”. A PortPrototypeBlueprint does not play any role in the formal description of any SwComponentType or related model artifacts (see also [TPS_STDT_0044]).

Class	**PortPrototypeBlueprint**			
Package	M2::AUTOSARTemplates::StandardizationTemplate::BlueprintDedicated::PortPrototypeBlueprint			
Note	This meta-class represents the ability to express a blueprint of a PortPrototype by referring to a particular PortInterface. This blueprint can then be used as a guidance to create particular PortPrototypes which are defined according to this blueprint. By this it is possible to standardize application interfaces without the need to also standardize software-components with PortPrototypes typed by the standardized PortInterfaces. **Tags:** atp.recommendedPackage=PortPrototypeBlueprints			
Base	ARElement,ARObject,AtpBlueprint,AtpClassifier,AtpFeature,AtpStructureElement,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
initValue	PortPrototypeBlueprintInitValue	*	aggr	This specifies the init values for the dataElements in the particular PortPrototypeBlueprint.
interface	PortInterface	1	ref	This is the interface for which the blueprint is defined. It may be a blueprint itself or a standardized PortInterface

[1] AUTOSAR does not standardize init values for application interfaces, but it is supported for vendor internal use.

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

Table 4.1: PortPrototypeBlueprint

Class	**PortPrototypeBlueprintInitValue**			
Package	M2::AUTOSARTemplates::StandardizationTemplate::BlueprintDedicated::PortPrototypeBlueprint			
Note	This meta-class represents the ability to express init values in PortPrototypeBlueprints. These init values act as a kind of blueprint from which for example proper ComSpecs can be derived.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
dataPrototype	AutosarDataPrototype	1	ref	This is the data prototype for which the init value applies **Tags:** xml.sequenceOffset=30
value	ValueSpecification	1	aggr	This is the init value for the particular data prototype. **Tags:** xml.sequenceOffset=40

Table 4.2: PortPrototypeBlueprintInitValue

As an AUTOSAR model taken for downstream model handling (e.g. generation of an RTE) requires the usage of complete `PortInterface`s it is necessary to derive an "actual" `PortInterface` out of a blueprinted `PortInterface` defined in the standardization process.

[TPS_STDT_0008] Compatibility of `PortPrototype` with Blueprint ⌈ [constr_2526], [constr_2527], [constr_2528] and [constr_2529] apply for the compatibility of `PortPrototype`s and `PortPrototypeBlueprint`s ⌋*(RS_STDT_0017)*

[constr_2526] `PortInterfaces` need to be compatible to the blueprints ⌈ `PortInterfaces` shall be compatible to their respective blueprints according to the compatibility rules. ⌋

[constr_2527] Blueprints shall live in package of a proper category ⌈ As explained in detail in the [10], model artifacts (in this case `PortPrototypeBlueprint` and incompletely specified `PortInterface`s) created for the purpose of becoming blueprints shall reside in an `ARPackage` of category `BLUEPRINT`. ⌋

[constr_2528] `PortPrototype`s shall not refer to blueprints of a `PortInterface` ⌈ A port `PortPrototype` shall not reference a `PortInterface` which lives in a package of category BLUEPRINT. ⌋

[constr_2529] `PortPrototypeBlueprint`s and derived `PortPrototype`s shall reference proper `PortInterfaces` ⌈ A `PortPrototypeBlueprint` may reference a blueprint of `PortInterface`. According to [constr_2570], a system description shall not contain blueprints. Therefore the reference to the `PortInterface` may need to be rewritten when a `PortPrototype` is derived from the blueprint.

In this case the PortInterface referenced by the derived PortPrototype shall be compatible to the PortInterface (which is a blueprint) referenced by the PortPrototypeBlueprint.

According to [constr_2526] this can be ensured if the PortInterface referenced by the PortPrototypeBlueprint is the blueprint of the PortInterface referenced by the respective PortPrototype. ⌋

Note that [constr_2529] is obviously also fulfilled if the PortPrototypeBlueprint and the derived PortPrototype reference a STANDARD PortInterface (which lives in a ARPackage of category "STANDARD").

4.15 Blueprinting PortInterface

[TPS_STDT_0009] Blueprinting PortInterface ⌈ PortInterface can be blueprinted. ⌋*(RS_STDT_0026)*

[constr_2500] PortInterfacess shall be of same kind ⌈ Both objects (PortInterfaces) referenced by a blueprint mapping for port interfaces (represented by BlueprintMapping) shall be of the same kind (e.g. both shall be SenderReceiverInterfaces). In other words both interfaces shall be instances of the same meta class. ⌋

Note that [constr_2500] is a special case of [constr_2566].

4.16 Blueprinting SwBaseType

[TPS_STDT_0022] Blueprinting SwBaseType ⌈ SwBaseType can be blueprinted. ⌋*(RS_STDT_0029)*

4.17 Blueprinting SwComponentType

[TPS_STDT_0024] Blueprinting SwComponentType ⌈ SwComponentType can be blueprinted. ⌋*(RS_STDT_0011, RS_STDT_0012)*

[constr_2568] SwComponentTypes shall be of same kind ⌈ Both objects (SwComponentTypes) referenced by a blueprint mapping for port interfaces (represented by BlueprintMapping) shall be of the same kind (e.g. both shall be AtomicSwComponentTypes). In other words both components shall be instances of the same meta class. ⌋

Note that [constr_2568] is a special case of [constr_2566].

5 Keywords

[TPS_STDT_0012] Defining Keywords ⌈ The meta-class `KeywordSet` can be used to define sets of `Keyword`s. The purpose of a `Keyword` is to contribute parts of names for AUTOSAR model elements. ⌋*(RS_STDT_0008, RS_STDT_0005)*

Keywords are referenced to be part of name pattern as specified in Chapter 3.4.

As an example, the `shortName` “CmftMngt” is composed out of two `Keyword`s with the `abbrName` “Cmft” and “Mngt”.

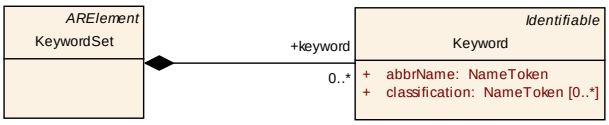

Figure 5.1: Keyword and KeywordSet

The meta-class `Keyword` is derived from `Identifiable`. The attributes of `Identifiable` shall be applied as follows.

`shortName` represents the unique name of the keyword. In the example above it would be “Cmft”. Note that this is used only for identifying the keyword. The contributed name part is taken from `abbrName`.

`longName` represents the long form of the keyword, typically its an unabbreviated technical term. In the example above it would be “Comfort”.

`desc` represents the definition of the keyword in terms of a verbal description allowing to identify whether the keyword applies for a specific case. In the example above the description would be “This keyword is used to express something as comfortable or convenient”.

`introduction` represents a verbal description of a use case. This can be used for additional explanations or examples.

Class	**KeywordSet**			
Package	M2::AUTOSARTemplates::StandardizationTemplate::Keyword			
Note	This meta–class represents the ability to collect a set of predefined keywords. **Tags:** atp.recommendedPackage=KeywordSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
keyword	Keyword	*	aggr	This is one particular keyword in the keyword set.

Table 5.1: KeywordSet

Class	**Keyword**			
Package	M2::AUTOSARTemplates::StandardizationTemplate::Keyword			
Note	This meta-class represents the ability to predefine keywords which may subsequently be used to construct names following a given naming convention, e.g. the AUTOSAR naming conventions. Note that such names is not only shortName. It could be symbol, or even longName. Application of keywords is not limited to particular names.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
abbrName	NameToken	1	attr	This attribute specifies an abbreviated name of a keyword. This abbreviation may e.g. be used for constructing valid shortNames according to the AUTOSAR naming conventions. Unlike shortName, it may contain any name token. E.g. it may consist of digits only.
classification	NameToken	*	attr	This attribute allows to attach classification to the Keyword such as MEAN, ACTION, CONDITION, INDEX, PREPOSITION

Table 5.2: Keyword

Note that the attribute `classification` depends on the applied naming convention. For example, the values should be according to table 2 of [11] such as `Action-PhysicalType`, `Condition-Qualifier`, `Index`, `Mean-Environment-Device`, `Preposition`.

Listing 5.1 illustrates an example how to use `Keyword`. More elaborate usage can be seen in [4].

Listing 5.1: example for keywords

```
<?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">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>AISpecification</SHORT-NAME>
          <AR-PACKAGES>
            <AR-PACKAGE>
              <SHORT-NAME>KeywordSets</SHORT-NAME>
              <ELEMENTS>
                <KEYWORD-SET>
                  <SHORT-NAME>KeywordList</SHORT-NAME>
                  <KEYWORDS>
                    <KEYWORD>
                      <SHORT-NAME>Cmft</SHORT-NAME>
                      <LONG-NAME>
                        <L-4 L="EN">Comfort</L-4>
```

```
                    </LONG-NAME>
                    <DESC>
                      <L-2 L="EN">comfort. this keyword is used to
                          express something as comfortable or convenient</
                          L-2>
                    </DESC>
                    <ABBR-NAME>Cmft</ABBR-NAME>
                    <CLASSIFICATIONS>
                      <CLASSIFICATION>Condition-Qualifier</CLASSIFICATION
                          >
                    </CLASSIFICATIONS>
                  </KEYWORD>
                </KEYWORDS>
              </KEYWORD-SET>
            </ELEMENTS>
          </AR-PACKAGE>
        </AR-PACKAGES>
      </AR-PACKAGE>
    </AR-PACKAGES>
  </AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>
```

6 Upcoming issues

[TPS_STDT_0038] Life Cycle Support ⌈ With the upcoming life cycle model, STDT will also able to express information about the state of the blueprints by references from within the life cycle info set added by this model. ⌋*(RS_STDT_0016)*

[TPS_STDT_0043] Blueprinting `LifeCycleDefinitionGroups` ⌈ With the upcoming life cycle model, STDT is also able to define blueprints of `LifeCycleDefinitionGroups`. This will look similar to the other blueprinting approaches specified in chapter 4 ⌋*(RS_STDT_0025)*

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 ([12]).

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 Change History

B.1 Change History R4.0.3

B.1.1 Added Constraints

Number	Heading
[constr_2500]	PortInterfaces shall be of same kind
[constr_2526]	PortInterfaces need to be compatible to the blueprints
[constr_2527]	Blueprints shall live in package of a proper category
[constr_2528]	PortPrototypes shall not refer to blueprints of a PortInterface
[constr_2529]	PortPrototypeBlueprints and derived PortPrototypes shall reference proper PortInterfaces
[constr_2540]	Tagged text category
[constr_2542]	Compatibility of introduction of blueprint and blueprinted element
[constr_2543]	Specify a name pattern in blueprints
[constr_2546]	References from Blueprint to Blueprint need to be replaced in derived objects
[constr_2553]	shortName shall follow the pattern defined in the Blueprint
[constr_2554]	Derived objects shall match the blueprints
[constr_2555]	Derived objects may have more attributes than the blueprints
[constr_2556]	No Blueprint Motivated VariationPoints in AUTOSAR Descriptions
[constr_2563]	BswModuleDescription blueprints should not have a BswModuleBehavior
[constr_2564]	VariationPoint in Blueprints of PackageableElements
[constr_2565]	Trace shall not be nested
[constr_2566]	Blueprintmapping shall map appropriate elements
[constr_2568]	SwComponentTypes shall be of same kind
[constr_2569]	Purely Bluprint Motivated VariationPoints
[constr_2570]	No Blueprints in system descriptions
[constr_2571]	Outgoing references from Blueprints

Table B.1: Added Constraints in 4.0.3

B.1.2 Added Specification Items

Number	Heading
[TPS_STDT_0037]	Port Direction
[TPS_STDT_0038]	Life Cycle Support
[TPS_STDT_0040]	Influence of ECUC
[TPS_STDT_0041]	Constraints may be Violated in Blueprints
[TPS_STDT_0042]	namePattern for short names of TraceableText in Template Documents
[TPS_STDT_0043]	Blueprinting LifeCycleDefinitionGroups
[TPS_STDT_0044]	Transferring VariationPoint
[TPS_STDT_0045]	Transferring Objects in General
[TPS_STDT_0046]	Configuration dependent properties
[TPS_STDT_0047]	Ignore Blueprint Attributes
[TPS_STDT_0048]	Express Decisions when Deriving Objects
[TPS_STDT_0049]	Blueprinting Enumerators
[TPS_STDT_0050]	namePattern for AUTOSAR delivered Files
[TPS_STDT_0051]	Handling references when deriving objects from blueprints

Table B.2: Added Specification Items in 4.0.3