Document Title	AUTOSAR Feature Model Exchange Format Requirements
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	605

Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.3.1

Document Change History

Date	Release	Changed by	Description
2017-12-08	4.3.1	AUTOSAR Release Management	Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	Editorial changes
2015-07-31	4.2.2	AUTOSAR Release Management	Editorial changes
2014-10-31	4.2.1	AUTOSAR Release Management	Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	Editorial changes
2013-03-15	4.1.1	AUTOSAR Administration	Initial release

Disclaimer

Table of Contents

Bibliography

[1] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

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

1 Introduction

1.1 Document Conventions

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([1]).

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([1]).

1.2 Requirements Tracing

The following table references the uses cases specified in *this document* and links to the fulfillments of these.

Requirement	Description	Satisfied by
[UC_FMDT_00001]	Overall Workflow	[RS_FMDT_00001] [RS_FMDT_00002] [RS_FMDT_00013]
[UC_FMDT_00002]	Exchange of Feature Models	[RS_FMDT_00001] [RS_FMDT_00002] [RS_FMDT_00013]
[UC_FMDT_00003]	Characteristics of Features	[RS_FMDT_00005] [RS_FMDT_00006]
[UC_FMDT_00004]	Restrictions for Features	[RS_FMDT_00008]
[UC_FMDT_00005]	Complex Restrictions for Features	[RS_FMDT_00008]
[UC_FMDT_00006]	Relations among Features	[RS_FMDT_00008]
[UC_FMDT_00007]	Attributes for Features	[RS_FMDT_00009]
[UC_FMDT_00008]	Distributed development of Feature Models	[RS_FMDT_00011] [RS_FMDT_00012]
[UC_FMDT_00009]	Feature Models are optional	[RS_FMDT_00014]
[UC_FMDT_00010]	Define a Feature Configuration for a concrete product	[RS_FMDT_00003]
[UC_FMDT_00011]	Exchange of Feature Configurations	[RS_FMDT_00003]
[UC_FMDT_00012]	Documentation for Features	[RS_FMDT_00004]
[UC_FMDT_00013]	Multiplicity of Features	[RS_FMDT_00007]
[UC_FMDT_00014]	Link Feature Modeling and Variant Handling	[RS_FMDT_00010]
[UC_FMDT_00015]	Cooperative Feature Model Development	[RS_FMDT_00011] [RS_FMDT_00012]
[UC_FMDT_00016]	BindingTimes for Features	[RS_FMDT_00015] [RS_FMDT_00016]

Table 1.1: RequirementsTracing

2 Use Cases

[UC_FMDT_00001] Overall Workflow ⌈ An OEM develops an AUTOSAR model and a feature model with toolset A. Then both models are passed to the supplier for completion. The supplier then enhances the work with toolset B and passes it back to the OEM. The OEM then re-imports the AUTOSAR model. This may happen several times during development cycles.

Different engineering domains use different feature modeling tools for variant management because specific tools better cover the individual needs; hence toolsets A and B are expected to be different.

At several synchronization points during development, not only the solutions but also the feature descriptions need to be integrated. ⌋()

[UC_FMDT_00002] Exchange of Feature Models ⌈ An OEM develops an AUTOSAR model and a feature model. The feature model is actually maintained in an external tool. This may be because the OEMs toolchain does not include a variant management tool that directly supports the AUTOSAR feature model, or because corporate standards call for a specific tool that does not have native support for the AUTOSAR feature model format.

Note: the feature model is not changed by the supplier in this use case. ⌋()

[UC_FMDT_00003] Characteristics of Features ⌈ A Feature Model developer wants to express certain characteristics of features:

- For clarity, feature models have a hierarchical structure[1], which is interpreted as follows: a feature may only be included into a product if its parent feature is also included in the product.
- A feature is mandatory for a product. For example, a car must have a steering wheel. It should be noted that (in accordance with the hierarchical structure) this does not mean that the feature is present in every product. A mandatory feature is only (but then, always) included in a product if its parent feature is included there. For example, if a car has a radio, speakers are also mandatory.
- A feature is optional, that is, it may or may not be present in a product. For example, a radio or a sunroof is an optional feature.
- Two or more features are marked as alternative: exactly one of them must be present in a product. For example, a car may have either a diesel or a gasoline engine.

[1] The hierarchy in question is actually a tree structure, meaning that each element (except the topmost one) has exactly one parent.

- Two or more features are marked as *multipleFeatures*: at least one of them must be present (the lower limit is not zero because this is already covered by optional features). It is possible to select several features.

⌋()

[UC_FMDT_00004] Restrictions for Features ⌈ Sometimes, a hierarchy is not sufficient to express all constraints on a feature model.

For example, there are features that are country specific, such as the location of the steering wheel or the default setting of the speedometer. However, it is not desirable to make “country x” a high level feature and arrange all other features below that feature, because this would lead to unnecessary repetition.

Instead, it is easier to position the feature “country x” at an appropriate place in the feature tree, and refer to that feature from any location in the feature tree. ⌋()

An example for a feature model with country-specific restrictions is shown in Figure 2.1.

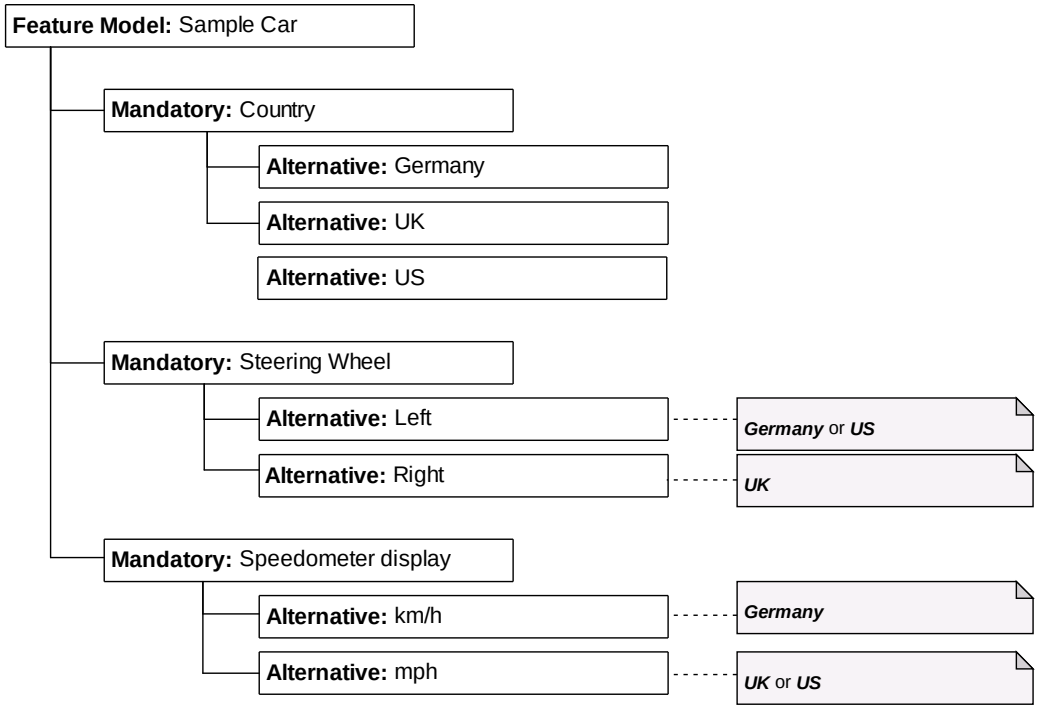

Figure 2.1: Example for Restrictions

[UC_FMDT_00005] Complex Restrictions for Features ⌈ A feature may be dependent on several other features. That is, it is only included in a product if all those other features are also included in the product. This cannot be expressed with the hierarchy proposed in Use Case [UC_FMDT_00003].

More complex types of restrictions may also apply. ⌋()

For example, Use Case [UC_FMDT_00004] could use more complex formulas of the form

(*Germany* or *US*) and not *UK*
UK and not (*Germany* or *US*)

(*UK* or *US*) and not *Germany*
Germany or not (*UK* or *US*)

[UC_FMDT_00006] Relations among Features ⌈ Similar to Use Cases [UC_FMDT_00004] and [UC_FMDT_00005], a feature model needs to express relations between feature where feature A requires or excludes the feature B.

This could also be expressed by putting a restriction (see Use Cases [UC_FMDT_00004] and [UC_FMDT_00005]) to feature B, but sometimes it is not possible to make such a change to feature B because its feature model cannot be changed. This may be because feature B is "owned" by a different party.

Furthermore, relations are often easier to understand or use than restrictions because they are simple keywords with a list of features, and not formulas.

Hence, feature A must be able to express its relationship with feature B. ⌋()

Figure 2.2 shows a feature model that follows the example in Figure 2.1 in Use Case [UC_FMDT_00004], but uses relations instead of restrictions. Note that the relations start at the country related features (*Germany*, *UK*, *US*), while the restrictions in the previous model are located at the *Steering Wheel* and *Speedometer* default features.

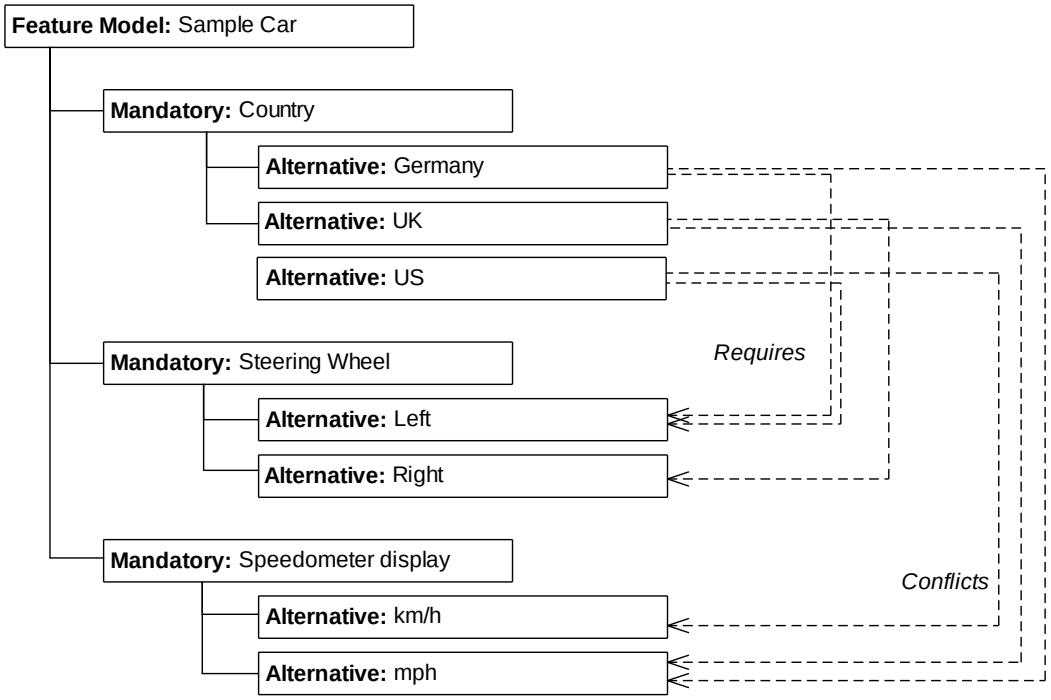

Figure 2.2: Example for Relations

Other examples for relations are *recommended for*, *discouraged for* and *impacts*.

[UC_FMDT_00007] Attributes for Features ⌈ A feature modeler wants to supply additional information to a feature, for example the maximum amount of bandwidth that the corresponding device may use.

Such information is useful if there are several options (*multipleFeatures* as in Use Case [UC_FMDT_00003]) where each feature corresponds to a separate device, but the total bandwidth that is available for these devices is limited by the characteristics of the bus. This would yield the restriction

$$(child1.bandwidth + child2.bandwidth + child3.bandwidth) < maximumBandwidth$$

⌋*()*

[UC_FMDT_00008] Distributed development of Feature Models ⌈ A feature model is developed by different entities. These entities may be different departments within the same company, or different companies altogether.

For example, an AUTOSAR software model (created by an OEM) that already has a feature model is integrated with another software model (created by a supplier) that comes with its own feature model.

As another example, consider two independent AUTOSAR software components, each of which comes with its own feature model. They must be integrated into a large AUTOSAR model which describes the whole system, and also contains its own feature model.

These examples are handled best if every party is able to edit and write their own file. The overall feature model is distributed over several files, or split into several different feature models that work together. ⌋*()*

[UC_FMDT_00009] Feature Models are optional ⌈ OEM A develops an AUTOSAR model with the help of feature models, and wants to include software components that are supplied by supplier B. However, B does not use feature modeling (or uses feature modeling but does not share its feature model for IP or contract reasons).

This does not preclude the use of AUTOSAR variant handling, which has been developed independent of feature modeling. ⌋*()*

[UC_FMDT_00010] Define a Feature Configuration for a concrete product ⌈ A feature model describes the features a product line and their interdependencies. Some of these features may be selectable. In contrast, a concrete product is described by a set of selected features. This set of selected features must satisfy the various constraints defined by the feature model.

To define the features of a concrete product, an OEM (or supplier) selects the subset of the features of the feature model. Furthermore, it must be checked that the feature selection adheres to the various constraints (see especially Use Cases [UC_FMDT_00003], [UC_FMDT_00004], [UC_FMDT_00005] and [UC_FMDT_00006]) as defined in the feature model.

This is repeated for every applicable product within the product line. That is, there can be multiple feature configurations. ⌋()

[UC_FMDT_00011] Exchange of Feature Configurations ⌈ An OEM defines a feature model for a product line, and then selects a number of feature configurations that define individual products as outlined in Use Case [UC_FMDT_00010]. Together with the feature model, these feature configurations are handed to a supplier to ensure that the concrete software works for the intended products (i.e., feature configurations). ⌋()

[UC_FMDT_00012] Documentation for Features ⌈ Experience has shown that it is a time consuming process to define the structure of a feature model (that is, which features are there, what is their hierarchical structure, what are their characteristics), establish relations between features and define which features are implemented by which system constants.

This is especially true if a feature model is created for a software product line which already exists. Typically, several people from different departments are involved in such a task.

Hence, the decisions that helped shaping the final version – the why? – of the feature model need to be documented. ⌋()

[UC_FMDT_00013] Multiplicity of Features ⌈ Features that are characterized as *multipleFeatures* in Use Case [UC_FMDT_00003] may supply a multiplicity constraint. This constraint restricts the number of features that may be included in a feature configuration.

For example, there may be 5 *multipleFeatures* features, but any feature configuration must include at least 2 and at most 4 such features. For example, a control panel may contain a number of switches, but there is space for at most four switches. ⌋()

[UC_FMDT_00014] Link Feature Modeling and Variant Handling ⌈ After creating a feature model, the developer needs to establish a link between the feature model and the variation points in the corresponding AUTOSAR model.

The relationship between features and variation points is not a one-to-one relationship. For example, one feature may influence several variation points, or one variation point may be influenced by more than one feature. ⌋()

[UC_FMDT_00015] Cooperative Feature Model Development ⌈ An OEM creates a feature model, exports it to an AUTOSAR feature model and transfers this model to a supplier. The supplier changes this model and hands it back to the OEM. The OEM then imports this model. ⌋*()*

[UC_FMDT_00016] BindingTimes for Features ⌈ A developer restricts the possible binding times for the implementation of a feature, for example to define that a feature should at least be implemented as *PreCompileTime*. This is described in the feature model.

Furthermore, there are two feature selections that are targeted at different customers: one customers wants a *PreCompileTime* implementation, and the other customer wants a *PostBuild* solution. This is described in the feature selection. ⌋*()*

3 Requirements

[RS_FMDT_00001] Support Product Lines ⌈

Type:	valid
Description:	A *Feature Model Exchange Format* should be able to express the basic functionality of a product line in terms of a set of related products which could have identical or shared features.
Rationale:	–
Use Case:	[UC_FMDT_00001],[UC_FMDT_00002]
Dependencies:	–
Supporting Material:	–

⌋*(UC_FMDT_00001, UC_FMDT_00002)*

[RS_FMDT_00002] Features ⌈

Type:	valid
Description:	A *Feature Model Exchange Format* should be able to express the basic functionality of a product in terms of features.
Rationale:	–
Use Case:	[UC_FMDT_00001],[UC_FMDT_00002]
Dependencies:	–
Supporting Material:	–

⌋*(UC_FMDT_00001, UC_FMDT_00002)*

[RS_FMDT_00003] Feature Selection ⌈

Type:	valid
Description:	A *Feature Model Exchange Format* should provide a feature selection mechanism that defines the feature set of a concrete product.
Rationale:	–
Use Case:	[UC_FMDT_00010],[UC_FMDT_00011]
Dependencies:	–
Supporting Material:	–

⌋*(UC_FMDT_00010, UC_FMDT_00011)*

[RS_FMDT_00004] Features should have names ⌈

Type:	valid
Description:	A *Feature Model Exchange Format* should be able to name and describe a feature.
Rationale:	–
Use Case:	[UC_FMDT_00012]
Dependencies:	[RS_FMDT_00003]
Supporting Material:	–

⌋(*UC_FMDT_00012*)

[RS_FMDT_00005] Feature Decomposition ⌈

Type:	valid
Description:	A *Feature Model Exchange Format* should be able to decompose a feature into subfeatures.
Rationale:	–
Use Case:	[UC_FMDT_00003]
Dependencies:	[RS_FMDT_00003]
Supporting Material:	–

⌋(*UC_FMDT_00003*)

[RS_FMDT_00006] Characteristics of Subfeatures ⌈

Type:	valid
Description:	Subfeatures should have different characteristics, for example “Mandatory”, “Optional”, and “Alternative”.
Rationale:	–
Use Case:	[UC_FMDT_00003]
Dependencies:	[RS_FMDT_00002], [RS_FMDT_00005]
Supporting Material:	–

⌋(*UC_FMDT_00003*)

[RS_FMDT_00007] Multiplicity of Features ⌈

Type:	valid
Description:	Features should be able to express a multiplicity. This is only relevant for the *multipleFeatures* type composition mentioned in Use Case [UC_FMDT_00013]. Mandatory, optional and alternative features do not have multiplicities.
Rationale:	–
Use Case:	[UC_FMDT_00013]
Dependencies:	[RS_FMDT_00002], [RS_FMDT_00007]
Supporting Material:	–

⌋(*UC_FMDT_00013*)

[RS_FMDT_00008] Relationships between features ⌈

Type:	valid
Description:	Features should be able to express different relationship w.r.t. to other features, such as “required”, “excluded”, and “impacted”.
Rationale:	–
Use Case:	[UC_FMDT_00004],[UC_FMDT_00005],[UC_FMDT_00006]
Dependencies:	[RS_FMDT_00002], [RS_FMDT_00005]
Supporting Material:	–

⌋(*UC_FMDT_00004*, *UC_FMDT_00005*, *UC_FMDT_00006*)

[RS_FMDT_00009] Attributes for features ⌈

Type:	valid
Description:	Feature should be able to have various attributes.
Rationale:	–
Use Case:	[UC_FMDT_00007]
Dependencies:	[RS_FMDT_00002]
Supporting Material:	–

⌋(*UC_FMDT_00007*)

[RS_FMDT_00010] Integration with AUTOSAR variant handling ⌈

Type:	valid
Description:	A *Feature Model Exchange Format* should be integrated with the existing AUTOSAR solution for variant handling.
Rationale:	–
Use Case:	[UC_FMDT_00014]
Dependencies:	–
Supporting Material:	–

⌋(*UC_FMDT_00014*)

[RS_FMDT_00011] Feature Model should be splitable ⌈

Type:	valid
Description:	A *Feature Model Exchange Format* should provide means to be able to be split the feature model into several different `ARXML` files.
Rationale:	–
Use Case:	[UC_FMDT_00008],[UC_FMDT_00015]
Dependencies:	–
Supporting Material:	–

⌋(*UC_FMDT_00008*, *UC_FMDT_00015*)

[RS_FMDT_00012] Distributed maintenance of Feature Models ⌈

Type:	valid
Description:	A *Feature Model Exchange Format* should provide means to distribute maintenance between different parties.
Rationale:	–
Use Case:	[UC_FMDT_00008],[UC_FMDT_00015]
Dependencies:	–
Supporting Material:	–

⌋(*UC_FMDT_00008*, *UC_FMDT_00015*)

[RS_FMDT_00013] Integration in AUTOSAR Methodology ⌈

Type:	valid
Description:	A *Feature Model Exchange Format* should be able to be integrated into the overall AUTOSAR Methodology.
Rationale:	–
Use Case:	[UC_FMDT_00001],[UC_FMDT_00002]
Dependencies:	–
Supporting Material:	–

⌋(*UC_FMDT_00001*, *UC_FMDT_00002*)

[RS_FMDT_00014] Feature Models are optional ⌈

Type:	valid
Description:	The usage of the *Feature Model Exchange Format* is optional in the scope of an AUTOSAR-compliant development cycle. This is similar to AUTOSAR variant handling; an AUTOSAR model that does not use variant handling is still a valid model.
Rationale:	–
Use Case:	[UC_FMDT_00009]
Dependencies:	–
Supporting Material:	–

⌋(*UC_FMDT_00009*)

[RS_FMDT_00015] Features may Specify Binding Times ⌈

Type:	valid
Description:	A feature may define an intended binding time that documents the binding time for the implementation of this feature. This attribute should be regarded as a hint.
Rationale:	–
Use Case:	[UC_FMDT_00016]
Dependencies:	–
Supporting Material:	–

⌋(*UC_FMDT_00016*)

[RS_FMDT_00016] Feature Selections may Specify Binding Times ⌈

Type:	valid
Description:	A feature selection may define a selected binding time that further refines the intended binding time from [RS_FMDT_00015]. This attribute should be regarded as a hint.
Rationale:	–
Use Case:	[UC_FMDT_00016]
Dependencies:	–
Supporting Material:	–

⌋(*UC_FMDT_00016*)

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 ([2]).

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 Meta-Model Tool The AUTOSAR Meta-Model Tool is the tool that generates different views (class tables, list of constraints, diagrams, XML Schema etc.) on the AUTOSAR meta-model.

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 Specification Element An AUTOSAR Specification Element is a named element that is part of an AUTOSAR specification. Examples: requirement, constraint, specification item, class or attribute in the meta model, methodology, deliverable, methodology activity, model element, bsw module etc.

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 Validation Tool A specialized AUTOSAR Tool which is able to check an AUTOSAR model against the rules defined by a profile.

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.

Life Cycle Life Cycle is the course of development/evolutionary stages of a model element during its life time.

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 creates an enhanced model out of an annotated model.

Profile Authoring Support Data Data that is used for efficient authoring of a profile. E.g. list of referable constraints, meta-classes, meta-attributes or other reusable model assets (blueprints)

Profile Authoring Tool A specialized `AUTOSAR Tool` which focuses on the authoring of profiles for data exchange points. It e.g. provides support for the creation of profiles from scratch, modification of existing profiles or composition of existing profiles.

Profile Compatibility Checker Tool A specialized `AUTOSAR Tool` which focuses on checking the compatibility of profiles for data exchange. Note that this compatibility check includes manual compatibility checks by engineers and automated assistance using more formal algorithms.

Profile Consistency Checker Tool A specialized `AUTOSAR Tool` which focuses on checking the consistency of profiles.

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 implemented 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`.