Document Title	AUTOSAR Feature Model Exchange Format
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	606

Document Status	published
Part of AUTOSAR Standard	Foundation
Part of Standard Release	R21-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2021-11-25	R21-11	AUTOSAR Release Management	• Changed all lower multiplicities in the meta-model to 0 and introduced constraints that define at which time which model elements need to be available. For details please refer to the ChangeDocumentation.
2020-11-30	R20-11	AUTOSAR Release Management	• Document moved to Foundation
2019-11-28	R19-11	AUTOSAR Release Management	• Editorial changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• 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	• Added [TPS_FMDT_00064]

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

References

[1] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate

[2] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[3] AUTOSAR Feature Model Exchange Format Requirements
AUTOSAR_RS_FeatureModelExchangeFormat

[4] Methodology for Classic Platform
AUTOSAR_TR_Methodology

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

1 Introduction and functional Overview

1.1 Variant Handling in AUTOSAR

Release 4 of AUTOSAR has added support for Variant Handling, which contributes two new aspects to the AUTOSAR metamodel as it was defined in release 3.

First, variation points have been introduced to AUTOSAR models. An AUTOSAR model with variation points describes a set of AUTOSAR models which have a common structure but differ at certain locations. A variant-free AUTOSAR model is created from such a model by binding the variation points, that is, by keeping some variations and discarding others.

Second, AUTOSAR defines means to express what constitutes a specific variant, for example which variation points are selected in an “economy” variant and which variation points are selected in a “luxury” variant. This is necessary because an AUTOSAR model with variation points may describe a very large number of variants, but few of them are actually used.

Variant Handling in AUTOSAR is described in chapter on variant handling of *the Generic Structure Template* [1].

1.2 The case for Feature Models

To summarize the previous section, AUTOSAR Variation points are intended to exchange information about where variation occurs in an AUTOSAR model, and to state what the relevant variants are. There are however two aspects that are not yet covered by this concept:

- Variation points are expressed on a rather low level. For example, the variants ’economy’ and ’luxury’ typically consist of a large number of variation points, but the fact that these variation points act conjointly is not explicitly visible in the model.
- There are dependencies between variation points. For example, ’economy’ and ’luxury’ variants are mutually exclusive, but this relationship is again not explicitly visible in the model. For variation points that are not PostBuild, this could be implemented by appropriately extending the formula language used for the conditions, although this would be difficult to maintain and would also intertwine two independent concepts, variation points and feature modeling. However, such an extension would not work for PostBuild because such variation points only use simple conditions.

The *Feature Model Exchange Format* which we are presenting in this document covers these additional aspects.

1.3 Sample Feature Model

An example for a feature model is shown in Figure 1.1.

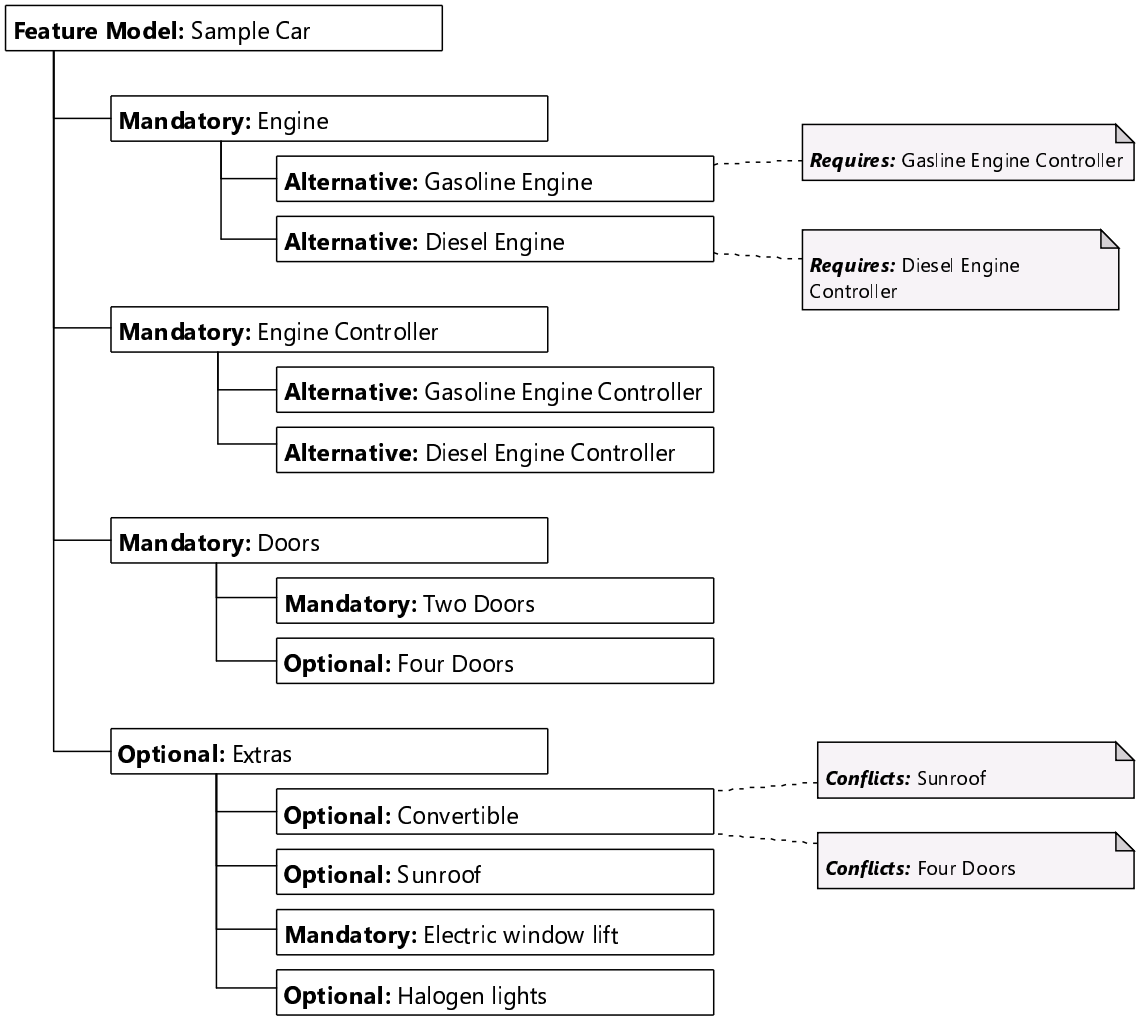

Figure 1.1: A sample Feature Model

1.4 Overview

1. Features reside in the “problem domain”. They are even independent from the implementation respectively the product architecture. They are much more abstract than variation points which reside in the “solution domain”. Features express common and variable characteristics of the finished product instead of annotating individual locations in a model. A feature model provides a high-level view of an AUTOSAR model with variations.
2. A feature model describes the dependencies between individual features. Examples for dependencies include hierarchical structuring of features, features that model alternatives, and features that require or prohibit other features.

3. An individual product can be described by selecting a set of features. Of course, such a selection has to obey the dependencies stated in the feature model. A mapping from features to variation points[1] specifies which variation points are affected by the selection.
4. The *Feature Model Exchange Format* establishes an efficient way to exchange feature models between different feature modeling tools.
5. Feature models will be optional in AUTOSAR. This means that feature models are an extension; it is still possible to develop and use AUTOSAR models that do not contain feature models.

1.5 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:

Please note that constraints are not supposed to be enforceable at any given time in an AUTOSAR workflow. During the development of a model, constraints may legitimately be violated because an incomplete model will obviously show inconsistencies.

However, at specific points in the workflow, constraints shall be enforced as a safeguard against misconfiguration.

The points in the workflow where constraints shall be enforced, sometimes also known as the "binding time" of the constraint, are different for each model category, e.g. on the classic platform, the constraints defined for software-components are typically enforced prior to the generation of the RTE while the constraints against the definition of an Ecu extract shall be applied when the Ecu configuration for the Com stack is created.

[1] More precisely, features will be mapped to values of system constants, which in turn control variation points.

For each document, possible binding times of constraints are defined and the binding times are typically mentioned in the constraint themselves to give a proper orientation for implementers of AUTOSAR authoring tools.

Let `AUTOSAR` be an example of a typical class table. 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.

Type: The type 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 attribute is aggregated in the class (`aggr` aggregation), an UML attribute in the class (`attr` primitive attribute), or just referenced by it (`ref` reference). Instance references are also indicated (`iref` instance reference) 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.

Please note that the chapters that start with a letter instead of a numerical value represent the appendix of the document. The purpose of the appendix is to support the explanation of certain aspects of the document and does not represent binding conventions of the standard.

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 ([2]).

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([2]).

[TPS_FMDT_00064] Usage of *Life Cycle* ⌈
Life Cycles in the FeatureModelExchangeFormat are described by making use of the Life Cycle Support as described in the *Generic Structure Template* [1].⌋ *(RS_FMDT_-00012)*

1.6 Requirements Tracing

The following table references the requirements specified in [3] and links to the fulfillments of these.

Requirement	Description	Satisfied by
[RS_FMDT_00001]	Support Product Lines	[TPS_FMDT_00004] [TPS_FMDT_00005] [TPS_FMDT_00006] [TPS_FMDT_00007] [TPS_FMDT_00008] [TPS_FMDT_00033] [TPS_FMDT_00043]
[RS_FMDT_00002]	Features	[TPS_FMDT_00002] [TPS_FMDT_00024] [TPS_FMDT_00035] [TPS_FMDT_00036] [TPS_FMDT_00042] [TPS_FMDT_00054] [TPS_FMDT_00055] [TPS_FMDT_00056]
[RS_FMDT_00003]	Feature Selection	[TPS_FMDT_00003] [TPS_FMDT_00009] [TPS_FMDT_00030] [TPS_FMDT_00032] [TPS_FMDT_00058] [TPS_FMDT_00059] [TPS_FMDT_00060]
[RS_FMDT_00004]	Features should have names	[TPS_FMDT_00039] [TPS_FMDT_00040] [TPS_FMDT_00052] [TPS_FMDT_00061] [TPS_FMDT_00062] [TPS_FMDT_00063]
[RS_FMDT_00005]	Feature Decomposition	[TPS_FMDT_00014] [TPS_FMDT_00030] [TPS_FMDT_00034] [TPS_FMDT_00036] [TPS_FMDT_00041]
[RS_FMDT_00006]	Characteristics of Subfeatures	[TPS_FMDT_00015] [TPS_FMDT_00016] [TPS_FMDT_00017] [TPS_FMDT_00018] [TPS_FMDT_00046]
[RS_FMDT_00007]	Multiplicity of Features	[TPS_FMDT_00012]
[RS_FMDT_00008]	Relationships between features	[TPS_FMDT_00019] [TPS_FMDT_00020] [TPS_FMDT_00021] [TPS_FMDT_00023] [TPS_FMDT_00030] [TPS_FMDT_00044] [TPS_FMDT_00045] [TPS_FMDT_00048] [TPS_FMDT_00049] [TPS_FMDT_00050] [TPS_FMDT_00057]
[RS_FMDT_00009]	Attributes for features	[TPS_FMDT_00051] [TPS_FMDT_00053]
[RS_FMDT_00010]	Integration with AUTOSAR variant handling	[TPS_FMDT_00025] [TPS_FMDT_00037] [TPS_FMDT_00038] [TPS_FMDT_00048] [TPS_FMDT_00049] [TPS_FMDT_00050] [TPS_FMDT_00057]
[RS_FMDT_00011]	Feature Model should be splitable	[TPS_FMDT_00047]
[RS_FMDT_00012]	Distributed maintenance of Feature Models	[TPS_FMDT_00033] [TPS_FMDT_00064]
[RS_FMDT_00013]	Integration in AUTOSAR Methodology	[TPS_FMDT_00033]
[RS_FMDT_00014]	Feature Models are optional	[TPS_FMDT_00001] [TPS_FMDT_00013]
[RS_FMDT_00015]	Features may Specify Binding Times	[TPS_FMDT_00054]
[RS_FMDT_00016]	Feature Selections may Specify Binding Times	[TPS_FMDT_00055]

Table 1.1: RequirementsTracing

2 Terminology

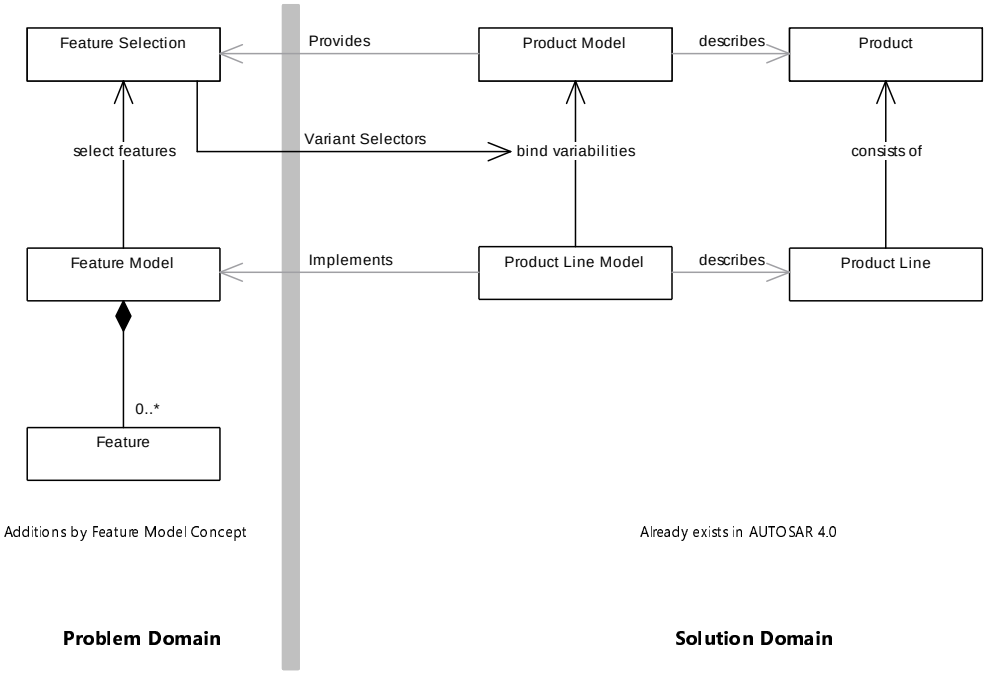

Figure 2.1: Overview of Feature Model Terminology

Figure 2.1 presents an overview of the terminology used for feature modeling. We define seven terms that are specific to the *Feature Model Exchange Format*, namely *Feature*, *Feature Selection*, *Feature Model*, *Product Model*, *Product Line Model*, *Product* and *Product Line*:

- **[TPS_FMDT_00002] Definition of *Feature*** ⌈
  A *Feature* describes an essential characteristic of a product. *Features* usually differentiate one product from similar products – in our context, features differentiate the individual products in a product line.⌋ *(RS_FMDT_00002)*

- **[TPS_FMDT_00003] Definition of *Feature Selection*** ⌈
  A *Feature Selection* is a set of *Features* that describes a specific product.

  A *Feature Selection* is always paired with a *Feature Model*. All dependencies and relations that are defined in the *Feature Model* shall be obeyed.⌋ *(RS_FMDT_00003)*

- **[TPS_FMDT_00004] Definition of *Feature Model*** ⌈
  A *Feature Model* describes the available features of a product line and their interrelations / interdependencies. In other words, a *Feature Model* describes a *Product Line Model* in the problem domain.

  For example, a car may have either a gasoline or a diesel engine, so the features *Gasoline* and *Diesel* are alternatives. A seven-seat configuration of a car might require air conditioning, so the feature *Seven Seats* requires the feature *Air Conditioning*.

A *Feature Model* is usually paired with a Product Line Model.⌋(RS_FMDT_00001)

- **[TPS_FMDT_00005] Definition of *Product Model*** ⌈
  A *Product Model* describes a product. A *Product Model* does not contain variation points anymore (except for *PostBuild* variation points). It is derived from the *Product Line Model* by "binding" the variation points.

  In AUTOSAR, a *Product Model* is a collection of M1 artifacts that describes a particular *Product*.

  Except for *PostBuild* variation points, The *Product Model* conforms to the Pure Metamodel as defined in the chapter on variant handling in the *Generic Structure Template* [1].⌋(RS_FMDT_00001)

- **[TPS_FMDT_00006] Definition of *Product Line Model*** ⌈
  A *Product Line Model* is similar to a *Product Model*, but contains variation points of all binding times. A *Product Model* is created out of a *Product Line Model* by keeping certain variations, and discarding others (i.e., binding). The only variation points that are still allowed in a *Product Model* are *PostBuild* variation points.

  In the context of feature modeling, this selection process is steered by a *Feature Selection* (more precisely, the variant selection Process (binding) is controlled by variant selectors (`SwSystemconst`) who's values in turn may be derived from Feature selection).

  In AUTOSAR, A *Product Line Model* is a collection of M1 artifacts which describe a set of *Product Models* with common characteristics. A *Product Line Model* is an instance of an extended metamodel as defined in the chapter on variant handling in the *Generic Structure Template* [1].⌋(RS_FMDT_00001)

- **[TPS_FMDT_00007] Definition of *Product*** ⌈
  A *Product is* an artifact that is the outcome of some type of process, for example a software that runs on one or more ECUs.

  In AUTOSAR, a *Product* is a collection of M0 artifacts. In our terminology, it is described by a *Product Model* on M1 Level.⌋(RS_FMDT_00001)

- **[TPS_FMDT_00008] Definition of *Product Line*** ⌈
  A *Product Line* is a collection of *Product*s that are related. A *Product Line* usually consists of a set of *Product*s that have a certain amount of common aspects and a number of aspects that differentiate the individual *Products* from each other.

  In our terminology, a *Product Line* is described by a *Product Line Model*, which in turn is described by a *Feature Model*.⌋(RS_FMDT_00001)

A note on the terminology

In this section, we define several terms that already have a meaning in AUTOSAR or elsewhere. This is especially true for the terms *Feature* and *Product*. A more concise definition might be *AUTOSAR Featuremodel Feature* or *AUTOSAR Featuremodel Product*.

However, it is easy to see that readability of this document would be significantly impacted by such a choice of words. Terms such as Feature have been established in the literature on feature modeling for some time, so using a different term would not be helpful for anybody familiar with the field.

Hence, we have decided to go with the established terms despite the overlap with existing AUTOSAR terminology. Since these terms are only used in the context of feature modeling, it should always be clear which definition is intended.

2.1 Terminology from graph theory

In this document, we do occasionally use concepts from graph theory to provide formal descriptions for constraints. The following sections defines these terms.

- A directed[1] *graph* is a tuple (V, E) where $E \subseteq V \times V$. V are called the *nodes* of G, and E are called the *edges* of G.
- A *path* p in a graph $G = (V, E)$ is a sequence $p = v_1, v_2, \ldots, v_n$ of nodes with $v_i \in V$ and $\forall i \in \{1, \ldots n-1\} : (v_i, v_{i+1}) \in E$. p *starts* at v_1 and *ends* at v_n.
- A *circle* in a graph $G = (V, E)$ is a path $v_1, v_2, \ldots, v_n$ where $v_1 = v_n$.
- A *self loop* in a graph $G = (V, E)$ is an edge $(v, v) \in E$.
- An *isolated node* in a graph $G = (V, E)$ is a node $v \in V$ where $\neg\exists v' : ((v, v') \in E \vee (v', v) \in E)$. In other words, an isolated node is one that has no edges.
- A *tree* is a graph $G = (V, E)$ with *root* $r \in V$ that has the following properties:
  1. $\forall v \in V : \exists \,\text{path}\, p = \{r, v_2, \ldots, v\}$. In other words, for every node $v \in V$, there exists a path that starts at r and ends at v.
  2. $\neg\exists v \in V : (v, r) \in E$. In other words, the root node has no incoming edge.
  3. $\forall v \in V, v \neq r : \dot{\exists} v' \in V : (v', v) \in E$. In other words, any node that is not the root node has exactly one incoming edge.
  4. G has no circles and no self loops.
  5. $\text{size}(V) = \text{size}(E) + 1$.

  Note: Items 4 and 5 are a consequence of items 1, 2 and 3.

[1] In this document, we need only directed graphs, so we are using the term *graph* synonymous with *directed graph*.

3 Overview

An AUTOSAR feature model consists of three different structures[1]: the *feature model* itself, the *feature selection*, and finally the *feature map*.

3.1 Feature Model

A feature model (FMFeatureModel) consists of a number of features (FMFeature), which are organized hierarchically (FMFeatureDecomposition). That is, each feature may contain a number of subfeatures, which in turn may contain subfeatures of their own and so on. In other words: a feature model consists of one or more feature trees.

As a special case, it is possible to distribute (split) feature models over several AUTOSAR files. It is also possible to partition a large feature tree into subtrees.

Also, there may be interdependencies between features. For example, features may represent alternatives and are thus mutually exclusive (FMFeatureDecomposition), or a feature may require one or more other features, contradict other features (FMFeatureRelation), or features may include an expression that restricts their applicability (FMFeatureRestriction).

3.2 Feature Selection

A feature selection is a set of features that describe an actual product. For example, a specific car model is described by its set of features. This is implemented by a FMFeatureSelectionSet, which contains a number of FMFeatureSelections, each of which defines the state of a FMFeature within this particular feature selection.

A feature selection is said to be valid if all the restrictions and relations defined for its features as well as the hierarchical structure of the feature model are obeyed.

Feature selections are handled separately from feature models because there usually are many different feature selections for a single feature model. For example, different cars are represented by different feature selections.

3.3 Feature Map

In AUTOSAR Variant Handling, variation points are controlled by system constants. An expression that is based on system constants is used to determine whether a particular

[1] Many concepts described in this document are adapted from *Generative Programming: Methods, Tools, and Applications*, Krzysztof Czarnecki and Ulrich W. Eisenecker, ACM Press/Addison-Wesley Publishing Co, 2000

variation point is ‘on’ or ‘off’. One consequence is that the same system constant may be used to control several variation points.

Hence, features cannot be mapped directly to variation points, but need to choose values for the system constants which are used in the variation points’ expressions.

This is done by feature maps (FMFeatureMap). In a nutshell, each element of a feature map (FMFeatureMapElement) contains a set of conditions (FMFeatureMapCondition) that are based on features and feature attributes and a set of assertions (FMFeatureMapAssertion) that are based on features and system constants. If any of the conditions and all of the assertions evaluate to *true*, then the mapping lists a number of system constants and chooses values for them.

4 Feature Model

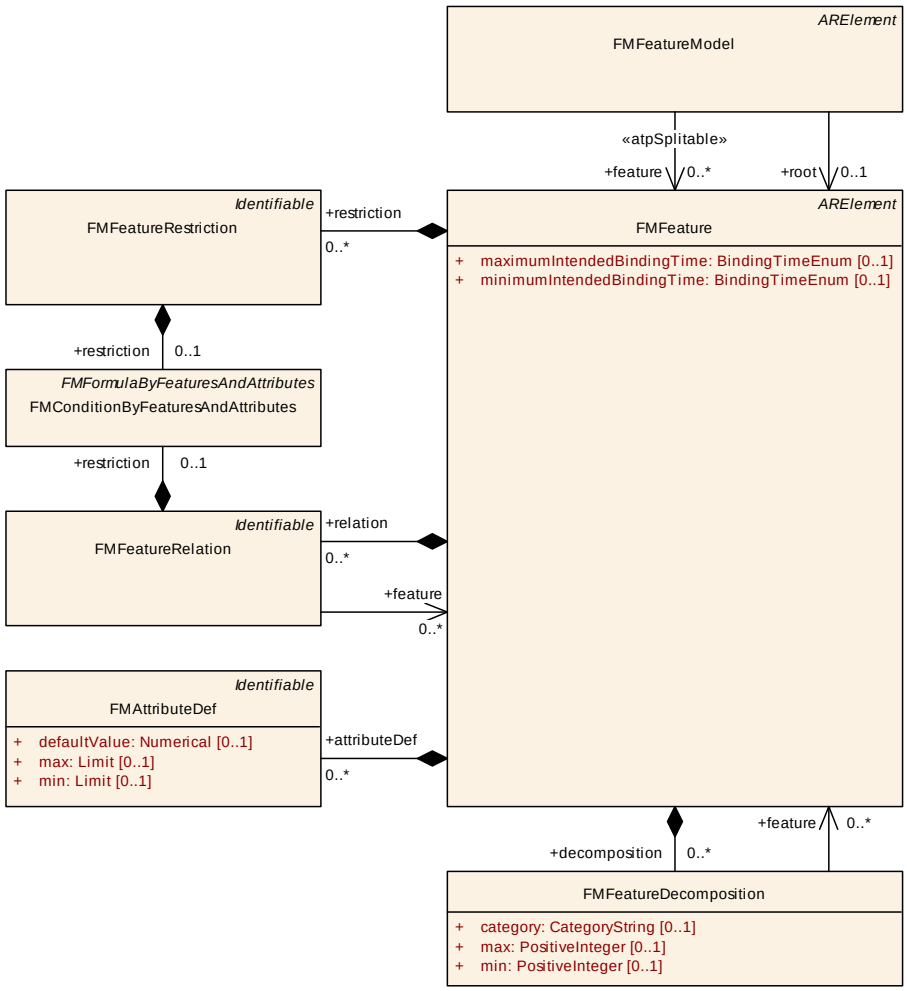

Figure 4.1: Class `FMFeatureModel`

4.1 Class `FMFeatureModel`

Class	**FMFeatureModel**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	A Feature model describes the features of a product line and their dependencies. Feature models are an optional part of an AUTOSAR model. **Tags:**atp.recommendedPackage=FMFeatureModels			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**FMFeatureModel**			
feature	FMFeature	*	ref	"feature" holds the list of features of the feature model. No FMFeature may be contained twice in this list. Also, each FMFeature may be contained on only one feature model. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=feature
root	FMFeature	0..1	ref	The features of a feature model define a tree. The attribute root points to the root of this tree.

Table 4.1: FMFeatureModel

[TPS_FMDT_00043] Purpose of `FMFeatureModel` ⌈A `FMFeatureModel` describes the available features of a product line, as defined in [TPS_FMDT_00004].⌋*(RS_FMDT_00001)*

A feature model is implemented by the class `FMFeatureModel`. As `FMFeatureModel` is an `ARElement`, an AUTOSAR model may contain any number of feature models, including *zero*.

[TPS_FMDT_00013] Feature Models are optional ⌈An AUTOSAR model that does not contain a feature model is still a valid AUTOSAR model.⌋*(RS_FMDT_00014)*

Especially, feature models may be empty, i.e., contain no features.

[TPS_FMDT_00001] Feature Models may be empty ⌈A `FMFeatureModel` may have zero references to `FMFeature` elements in the role `feature`.⌋*(RS_FMDT_00014)*

If an AUTOSAR model contains more than one feature model, then these feature models may interact with each other in two ways. First, feature models may use other feature models as sub-models, as defined in Section 4.4.4. Second, restrictions (4.5) and relations (4.6) between features may refer to features that are defined in other feature models.

4.1.1 Reference `feature`

Each `FMFeatureModel` contains a number of `FMFeature` elements in the role `feature`. These elements represent the *features* of the feature model.

[TPS_FMDT_00035] Definition of *Features* of a `FMFeatureModel` ⌈Let F be a `FMFeatureModel` and let $\{f_1, f_2, \ldots, f_n\}$ be the set of `FMFeature`s that are referenced from F in the role `feature`. Then $\{f_1, f_2, \ldots, f_n\}$ are the *features of* F.⌋*(RS_FMDT_00002)*

A `FMFeature` can only be part of a single `FMFeatureModel`:

[constr_5007] `FMFeature` shall only be referenced from one `FMFeatureModel` in the role `feature` ⌈Let f be a `FMFeature`, and F,F' be `FMFeatureModel`s where F references f in the role `feature`, and F' also references f in the role `feature`. Then $F = F'$.⌋*()*

Obviously, a FMFeatureModel shall not contain the same feature twice.

[constr_5019] FMFeatureModel shall not contain the same FMFeature twice ⌈Let F be a FMFeatureModel, and let f, f' be FMFeatures that are referenced from F in the role feature. Then $f \neq f'$.⌋*()*

On the other hand, there are no "isolated" features; every FMFeature is part of a FMFeatureModel.

[constr_5020] Every FMFeature shall be contained in a FMFeatureModel ⌈For every FMFeature f, there shall be a FMFeatureModel that refers to f in the role feature.⌋*()*

Constraint [constr_5020] makes sure that there are no "standalone" features, which would be technically possible because FMFeature is an ARElement, but is not useful in this context.

Finally, feature models can be distributed over several physical ARXML files if necessary.

[TPS_FMDT_00047] Feature models are splitable ⌈The relation feature has the stereotype ≪atpSplitable≫. That is, a FMFeatureModel may be distributed over several ARXML files.⌋*(RS_FMDT_00011)*

4.1.2 Reference root

As the features of a feature model are organized in a tree structure (see Section 4.4), there is exactly one feature that sits at the top of the tree. The feature model has an extra reference root which points to that feature.

root is not strictly necessary because it would be possible to infer the root feature from the hierarchical structure of a feature model (see Section 4.4). However, root is included for convenience and to assist tools in checking the integrity of the model.

[TPS_FMDT_00036] Definition of *Root Feature* of a FMFeatureModel ⌈Let F be a FMFeatureModel that refers to a FMFeature f in the role root. Then f is called the *root feature* of F.⌋*(RS_FMDT_00002, RS_FMDT_00005)*

We need to define two constraints for the root feature. First, if the feature model is not empty – that is, it has features – then one feature shall be the root feature:

[constr_5009] Root feature shall be present if and only if the feature model is not empty ⌈If a FMFeatureModel refers to one or more FMFeature elements in the role feature, then exactly one of them shall be referenced by FMFeatureModel in the role root.

On the contrary, if FMFeatureModel does not refer to any FMFeatures in the role feature, then root shall be empty.⌋*()*

Second, the root feature of a feature model shall be one of its own features:

[constr_5008] If present, the root feature shall be part of the feature model ⌈Let r be the FMFeature referenced from FMFeatureModel in the role root, and $\{f_1, f_2, \ldots, f_n\}$ the set of features referenced from the same FMFeatureModel in the role feature.

Then the following condition shall hold: $r \in \{f_1, f_2, \ldots, f_n\}$.⌋*()*

We will come back to the root feature later with constraint [constr_5022] where we require that the root feature points to the root of the feature tree, and [constr_5010] where we allow a feature to use the root feature (but only that) of another feature model as a subfeature.

4.2 Class FMFeature

Each FMFeatureModel consists of a number of FMFeatures, which in turn are organized in a hierarchical, tree-like structure. This hierarchy establishes a parent-child relation among features, where every feature that is not the root feature has exactly one parent, and any number of children, including zero.

Class	**FMFeature**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	A FMFeature describes an essential characteristic of a product. Each FMFeature is contained in exactly one FMFeatureModel. **Tags:**atp.recommendedPackage=FMFeatureModels			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
attributeDef	FMAttributeDef	*	aggr	This defines the attributes of the given feature.
decomposition	FMFeature Decomposition	*	aggr	Lists the sub-features of a feature.
maximum IntendedBinding Time	BindingTimeEnum	0..1	attr	Defines an upper bound for the binding time of the variation points that are associated with the FMFeature. This attribute is meant as a hint for the development process.
minimum IntendedBinding Time	BindingTimeEnum	0..1	attr	Defines a lower bound for the binding time of the variation points that are associated with the FMFeature. This attribute is meant as a hint for the development process.
relation	FMFeatureRelation	*	aggr	Defines relations for FMFeatures, for example dependencies on other FMFeatures, or conflicts with other FMFeatures. A FMFeature can only be part of a FMFeatureSelectionSet if all its relations are fulfilled.
restriction	FMFeatureRestriction	*	aggr	Defines restrictions for FMFeatures. A FMFeature can only be part of a FMFeatureSelectionSet if at least one of its restrictions evaluates to true.

Table 4.2: FMFeature

[TPS_FMDT_00042] Purpose of FMFeature ⌈A FMFeature describes an essential characteristic of a product, as defined in [TPS_FMDT_00002].⌋*(RS_FMDT_00002)*

A FMFeature aggregates the following elements:

FMFeatureDecomposition A *decomposition* defines how features are organized hierarchically. It also imposes certain constraints among features: there are *mandatory*, *optional*, *alternative* and *multiple*-features.

Feature decomposition is described in Section 4.4.

FMFeatureRestriction A *restriction* contains a formula that constrains the inclusion of a feature into a valid *feature selection* ([TPS_FMDT_00030]). There may also be more than one restriction. A feature may only be part of a valid feature selection if at least one its restrictions evaluates to *true*.

Feature restrictions are described in Section 4.5.

FMFeatureRelation A *relation* expresses constraints among features. A relation points from one feature to one or more other features and defines a relationship between these features. For example, relationships may be used to express that one feature *requires* another feature, or *conflicts* with several other features.

Feature relations are described in Section 4.6.

FMAttributeDef An *attribute* defines a numerical attribute of a feature. Attributes are used by restrictions (see Section 4.5) and feature maps (see Section 6 6.4). Features themselves define only the attribute and an optional default value; the actual value may be further refined in a feature selection (Section 5).

Feature attributes are described in Section 4.3.

`FMFeature` has two attributes, `maximumIntendedBindingTime` and `minimumIntendedBindingTime`, which specify the intended binding time for the variation points that are associated with this feature (see Section 4.2.2).

4.2.1 Name and Documentation of a Feature

[TPS_FMDT_00039] Name of a FMFeature ⌈The attribute `shortName` may be used to identify a feature. Furthermore, the attribute `longName` may be used to provide a human readable name for a `FMFeature`.⌋*(RS_FMDT_00004)*

[TPS_FMDT_00040] Description for a FMFeature ⌈The attributes `introduction` and `desc` may be used to provide a human readable description for a `FMFeature`.⌋ *(RS_FMDT_00004)*

As outlined in [1], `introduction` and `desc` are intended to be used as follows:

- `introduction` [TPS_GST_00103] contains introductory documentation about *how the feature may be used*.
- `desc` [TPS_GST_00100] contains a brief description about *what the feature is*.

The attributes `shortName`, `longName`, `introduction` and `desc` are not visible in Figure 4.1, but stem from the fact that `FMFeature` is based on `ARElement`, which in turn is based on `Identifiable` and `Referrable`.

4.2.2 Intended Binding Time

The class FMFeature contains two optional attributes minimumIntendedBindingTime and maximumIntendedBindingTime, which define lower and upper bounds for the intended binding time (binding times are explained in the AUTOSAR Methodology[4]) of the variation points that are associated with the FMFeature.

[TPS_FMDT_00054] Semantics of attributes minimumIntendedBindingTime and maximumIntendedBindingTime ⌈Let f be a FMFeature and V be the set of affected variation points of f as defined in [TPS_FMDT_00038]. Then the following conditions are implied for every variation point $v \in V$:

1. If the attribute minimumIntendedBindingTime exists and has value min, then $min \leq bindingtime(v)$.
2. If the attribute maximumIntendedBindingTime exists and has value max, then $bindingtime(v) \leq max$.

⌋*(RS_FMDT_00002, RS_FMDT_00015)*

[TPS_FMDT_00054] refers to the variation points that are associated with a FMFeature. This information is not available through a FMFeatureModel, but is defined in a FMFeatureMap (see Section 6). Hence, the attributes minimumIntendedBindingTime and maximumIntendedBindingTime can only be interpreted when a FMFeatureMap is also available.

[TPS_FMDT_00024] Attributes maximumIntendedBindingTime and minimumIntendedBindingTime are only a hint ⌈The values of maximumIntendedBindingTime and minimumIntendedBindingTime are only meant as a hint for the development process to guide the selection of correct variability implementation.⌋*(RS_FMDT_00002)*

4.3 Attributes of a Feature

Each FMFeature aggregates zero or more FMAttributeDef elements, each of which defines an attribute of a feature.

Class	**FMAttributeDef**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	This metaclass represents the ability to define attributes for a feature.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
defaultValue	Numerical	0..1	attr	This represents the default value of the attribute.
max	Limit	0..1	attr	Maximum possible value for the value of this attribute
min	Limit	0..1	attr	Minimum possible value for the value of this attribute

Table 4.3: FMAttributeDef

[constr_3657] Multiplicity of `FMAttributeDef.max` and `FMAttributeDef.min`
⌈For each `FMAttributeDef` the attributes `max` and `min` shall exist.⌋*()*

[TPS_FMDT_00051] Purpose of `FMAttributeDef` ⌈`FMAttributeDef` defines attributes for features. Each `FMAttributeDef` contains an optional `defaultValue` and defines limits for its value with the attributes `max` and `min`.⌋*(RS_FMDT_00009)*

[constr_5026] Semantics of attributes `max` and `min` in class `FMAttributeDef`
⌈The following conditions shall hold for all instances of the class `FMAttributeDef`:

- $\texttt{min} \leq \texttt{defaultValue} \leq \texttt{max}$ (`min` and `max` are both closed intervals)
- $\texttt{min} < \texttt{defaultValue} \leq \texttt{max}$ (`min` is an open interval, `max` is a closed interval)
- $\texttt{min} < \texttt{defaultValue} < \texttt{max}$ (`min` and `max` are both open intervals)
- $\texttt{min} \leq \texttt{defaultValue} < \texttt{max}$ (`min` is a closed interval, `max` is an open interval)

⌋*()*

Since `FMAttributeDef`s are `Identifiable`s, they have a `shortName` that can be used as the name of the attribute.

An example on how to use attributes is presented in Section 5.2.3.1.

4.4 Class `FMFeatureDecomposition`

Each `FMFeature` aggregates one or more `FMFeatureDecomposition` elements. A `FMFeatureDecomposition` contains references to other features, and thus establishes a hierarchical organization of features. This hierarchy imposes certain restrictions on `FMFeature`s, for example declares some features as optional or mutually exclusive. It may also connect one `FMFeatureModel` to another `FMFeatureModel` by referring to its `root` feature.

Class	**FMFeatureDecomposition**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	A FMFeatureDecomposition describes dependencies between a list of features and their parent feature (i.e., the FMFeature that aggregates the FMFeatureDecomposition). The kind of dependency is defined by the attribute category.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
category	CategoryString	0..1	attr	The category of a FMFeatureDecomposition defines the type of dependency that is defined by the FMFeature Decomposition. There are four different categories: MANDATORYFEATURE, OPTIONALFEATURE, ALTERNATIVEFEATURE, and MULTIPLEFEATURE.
feature	FMFeature	*	ref	The features that are affected by the dependency defined by the FMFeatureDecomposition.
max	PositiveInteger	0..1	attr	For a dependency of category MULTIPLEFEATURE, this defines the maximum number of features allowed.

▽

△

Class	**FMFeatureDecomposition**			
min	PositiveInteger	0..1	attr	For a dependency of category MULTIPLEFEATURE, this defines the minimum number of features allowed.

Table 4.4: FMFeatureDecomposition

[constr_3658] Multiplicity of `FMFeatureDecomposition.category` ⌈For each `FMFeatureDecomposition` the attribute `category` shall exist.⌋ *()*

[constr_3659] Multiplicity of `FMFeatureDecomposition.feature` ⌈For each `FMFeatureDecomposition` at least one reference in the role `feature` shall exist.⌋ *()*

[TPS_FMDT_00041] Purpose of `FMFeatureDecomposition` ⌈Each `FMFeature` aggregates zero or more `FMFeatureDecomposition` elements in the role `decomposition`. `FMFeatureDecomposition` thus establishes a hierarchical organization of `FMFeature`s.⌋ *(RS_FMDT_00005)*

A `FMFeature` that has no `FMFeatureDecomposition` is a *leaf* in the feature tree.

4.4.1 Constraints and Terminology for `FMFeatureDecomposition`

[TPS_FMDT_00014] Definition of *Parent Feature*, *Child Feature* ⌈Let f be a `FMFeature` which aggregates a `FMFeatureDecomposition` that references a `FMFeature` f' in the role `feature`. Then f is the *parent feature* of f', and f' is a *child feature* of f.⌋ *(RS_FMDT_00005)*

Each feature has at most one *parent* feature, but can have any number of child features, including zero. This is established by the fact that a feature model is organized as a tree, as determined by constraint [constr_5021] below.

[constr_5005] `FMFeature` shall not be referenced from more than one `FMFeatureDecomposition` ⌈Let f be a `FMFeature` that is referenced from a `FMFeatureDecomposition` in the role `feature`. Then no other `FMFeatureDecomposition` shall reference f in the role `feature`.⌋ *()*

Constraint [constr_5005] makes sure that every `FMFeature` has at most one parent feature (the number of child features is not limited for obvious reasons). This paves the way for the following definition of the underlying graph of a `FMFeatureModel`, which is in fact an underlying tree.

[TPS_FMDT_00034] Definition of *Underlying Graph* of a `FMFeatureModel` ⌈Let F be a `FMFeatureModel` and $\{f_1, f_2, \ldots, f_n\}$ be the set of `FMFeature`s that are referenced from F in the role `feature`.

Then the *underlying graph* of F is a graph $G = (V, E)$ where

$$V = \{f_1, f_2, \ldots, f_n\}$$

and

$$E = \{(f_i, f_j) \mid f_i \text{ is the parent feature of } f_j\}$$

⌋*(RS_FMDT_00005)*

[constr_5021] The underlying graph of a feature model shall be a tree. ⌈Let F be a FMFeatureModel and G be the underlying graph of F as defined in [TPS_FMDT_00034]. Then G shall be a tree. Hence, we also refer to G as the *underlying tree* of F.⌋*()*

[constr_5022] The root feature of a FMFeatureModel refers to the root of the underlying tree. ⌈Let F be a FMFeatureModel and G be the underlying tree of F as defined in [TPS_FMDT_00034]. Furthermore, let r be the FMFeature referred to by the root feature of the FMFeatureModel.

Then the node in G which corresponds to r is the root of the tree G.⌋*()*

4.4.2 Categories of Feature Decompositions

The attribute category of a FMFeatureDecomposition defines the semantics for the FMFeatures referenced in the role feature. We define four categories for FMFeatureDecomposition, namely MANDATORYFEATURE, OPTIONALFEATURE, ALTERNATIVEFEATURE and MULTIPLEFEATURE:

- **[TPS_FMDT_00015] MANDATORYFEATURE** ⌈All FMFeatures referenced in the role feature from a FMFeatureDecomposition that has the category MANDATORYFEATURE *shall* be present in a feature selection if and only if its parent FMFeature is included in the feature selection.⌋*(RS_FMDT_00006)*
- **[TPS_FMDT_00016] OPTIONALFEATURE** ⌈FMFeatures referenced in the role feature from a FMFeatureDecomposition that has the category OPTIONALFEATURE *may* be present in a feature selection if and only if its parent FMFeature is included in the feature selection.⌋*(RS_FMDT_00006)*
- **[TPS_FMDT_00017] ALTERNATIVEFEATURE** ⌈Exactly one of the FMFeatures referenced in the role feature from a FMFeatureDecomposition that has the category ALTERNATIVEFEATURE *shall* be present in a feature selection if and only if its parent FMFeature is included in the feature selection.⌋*(RS_FMDT_00006)*
- **[TPS_FMDT_00018] MULTIPLEFEATURE** ⌈One or more of the FMFeatures referenced in the role feature from a FMFeatureDecomposition that has the category MULTIPLEFEATURE *shall* be present in a feature selection if and only if its parent FMFeature is included in the feature selection. This is further constrained by the attributes min and max (see [TPS_FMDT_00012] and [constr_5013]).⌋*(RS_FMDT_00006)*

These definitions are formalized in [TPS_FMDT_00046].

[TPS_FMDT_00046] Semantics of FMFeatureDecomposition ⌈Let S be a set of FMFeatures and let $f, f_1, f_2, \ldots, f_n$ be FMFeatures where f is the parent feature for

$f_1, f_2, \ldots, f_n$. Furthermore, let d be the `FMFeatureDecomposition` that is aggregated by f in the role `decomposition` where $\{f_1, f_2, \ldots, f_n\}$ are all referenced from d in the role `feature`.

Based on the `category` of the `FMFeatureDecomposition` d, the following conditions are defined:

MANDATORYFEATURE
$$f \in S \Leftrightarrow |\{f_1, f_2, \ldots, f_n\} \cap S| = n$$

OPTIONALFEATURE
$$f \in S \Leftrightarrow 0 \leq |\{f_1, f_2, \ldots, f_n\} \cap S| \leq n$$

ALTERNATIVEFEATURE
$$f \in S \Leftrightarrow |\{f_1, f_2, \ldots, f_n\} \cap S| = 1$$

MULTIPLEFEATURE
$$f \in S \Leftrightarrow \min \leq |\{f_1, f_2, \ldots, f_n\} \cap S| \leq \max$$

⌋*(RS_FMDT_00006)*

Note that [TPS_FMDT_00046] does not *require* that the conditions are fulfilled. Only if S is a *valid feature selection* (see [TPS_FMDT_00030]), then all conditions have to be fulfilled. This is necessary because a feature selection may be incomplete, for example if features are selected in a step-by-step process where only the "final" feature selection fulfills all constraints.

4.4.3 Attributes `min` and `max`

If the optional attributes `min` and `max` are present, they restrict how many *multiple* features may be selected.

[TPS_FMDT_00012] Default values for attributes `min` and `max` of `FMFeatureDecomposition` ⌈If `min` and `max` are missing, then the values 1 (for `min`) and ∞ (for `max`) are assumed in [TPS_FMDT_00046].⌋*(RS_FMDT_00007)*

In other words, if `min` and `max` are not specified, then a valid feature selection shall contain at least one of the features, but there is no upper bound. Technically, ∞ in [TPS_FMDT_00012] translates to the maximum number that can be represented by `PositiveInteger`.

[constr_5013] Attributes `min` and `max` of `FMFeatureDecomposition` reserved for `category` MULTIPLEFEATURE ⌈The optional attributes `min` and `max` of `FMFeatureDecomposition` are only allowed to be present if the `category` of the `FMFeatureDecomposition` is `MULTIPLEFEATURE`.⌋*()*

4.4.4 Hierarchical decomposition of Feature Models

There is a special case where FMFeatureDecomposition may reference a feature in another FMFeatureModel. This is useful for hierarchical decomposition of FMFeatureModels. However, this is only allowed if the referenced feature is the root feature.

[constr_5010] FMFeatureDecomposition may refer to a root feature of another feature model, but only once. ⌈Let f_A be a FMFeature that is referenced by FMFeatureModel A in the role feature, but is also referenced from a FMFeatureDecomposition that is aggregated by a FMFeature f_B in the role decomposition.

Furthermore, let B be the FMFeatureModel that references f_B in the role feature with $A \neq B$. That is, f_A and f_B belong to different feature models.

Then *both* the following conditions shall hold:

1. f_A is referenced from A in the role root.
2. There is no other FMFeatureDecomposition (neither in B nor in any other FMFeatureModel) that references f_B in the role feature.

⌋()

The second condition in [constr_5010] is necessary to make sure that the overall structure of the combined feature models is still a tree (see also [TPS_FMDT_00034] and [constr_5021]).

4.4.5 Why use referencing for FMFeature instead of aggregation?

We could also have defined feature models such that FMFeatureModel aggregates a single FMFeature (the root feature), and FMFeatureModel then recursively aggregates other FMFeatures. With this approach, several of the constraints defined earlier in this section would have been unnecessary because this aggregation naturally forms a tree.

However, this approach would not work for the decomposition of feature models as described in Section 4.4.4. In this case, a FMFeatureDecomposition refers to the root of a different FMFeatureModel. This cannot easily be done by aggregation.

4.5 Class FMFeatureRestriction

The hierarchy established by FMFeatureDecomposition (see Section 4.4) covers many use cases for constraining the inclusion of a feature into a feature selection.

There are however circumstances where more elaborate constraints are necessary. FMFeatureDecomposition defines constraints for features which share the same *parent* features. For example, it may express that several features are *alternatives*

within the context of their parent (typically, a “Car” either contains a “Diesel” or a “Gasoline” engine, but not both), but it cannot express that a FMFeature depends on another FMFeature accross the tree, or contradicts a combination of two other features.

A FMFeature may aggregate a number of FMFeatureRestriction elements that further limit its inclusion in a feature selection. A FMFeatureRestriction aggregates a boolean[1] expression in the role restriction which constrains whether a particular feature is allowed to become part of a FMFeatureSelection.

More precisely, a feature may only become part of a feature selection if *at least one* of its restrictions evaluate to *true*. That is, all the restrictions are merged into a single boolean expression; the individual restrictions are combined by a $\vee$ operator. For simlicity, there are no priorities among the restrictions.

Class	**FMFeatureRestriction**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	Defines restrictions for FMFeatures. A FMFeature can only be part of a FMFeatureSelectionSet if at least one of its restrictions evaluate to true.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
restriction	FMConditionByFeatures AndAttributes	0..1	aggr	A formula that contains the actual restriction.

Table 4.5: FMFeatureRestriction

[TPS_FMDT_00045] Semantics of FMFeatureRestriction ⌈Let S be a feature selection for a FMFeatureModel, and f be a FMFeature with a set of FMFeatureRestrictions $\{R_1, R_2, \ldots, R_n\}$. Let $\{C_1, C_2, \ldots, C_n\}$ be the FMFormulaByFeaturesAndAttributes elements that are aggregated by R_i in the role restriction. Then the feature defines the following condition:

$$f \in S \Rightarrow C_1 = \text{true} \vee C_2 = \text{true} \vee \ldots \vee C_n = \text{true}$$

⌋*(RS_FMDT_00008)*

Note that [TPS_FMDT_00045] only *defines* a condition, but does not require that the condition is fulfilled by the feature selection. Only a *valid feature selection* (see [TPS_FMDT_00030]) requires that the condition is fulfilled.

The condition stated in [TPS_FMDT_00045] works only in one direction. Even if all conditions are *true*, a specific feature may still be left out of a valid feature selection. On the contrary, if the feature is contained in a valid feature selection, at least one restriction has to be true. This is different from a FMFeatureRelation, which may force a feature to be part of a valid feature selection (see Section 4.6).

We do not impose further constraints on the restrictions that can be used with FMFeatureRestriction. This means that it is in the responsibility of the creator[2] of the

[1] I.e., its value is interpreted as a boolean value.
[2] The creator of the restriction can be one or more people, or even a tool.

restriction to make sure that no circular dependencies or conflicts are introduced. For example, it is perfectly legal for a feature f to have the restriction $\neq f$, although this may not very useful because the feature can never be selected.

4.5.1 Identifying and documenting FMFeatureRestrictions

Because FMFeatureRestriction is based on Identifiable, it may contain the optional attributes shortName, introduction, and desc.

[TPS_FMDT_00062] Identifying FMFeatureRestrictions ⌈The attribute shortName can be used to distinguish relations in case a FMFeature aggregates several FMFeatureRestrictions.⌋*(RS_FMDT_00004)*

[TPS_FMDT_00063] Documenting FMFeatureRestrictions ⌈The attributes introduction and desc may be used to provide a human readable description for a FMFeatureRestriction.⌋*(RS_FMDT_00004)*

4.5.2 Example

Consider a feature f that has the following restriction:

$$f_1 \&\& f_2 \&\& f_3$$

This restriction defines that f may only be part of a feature selection if f_1, f_2 and f_3 are also included in that feature selection. This cannot be expressed with a feature tree because f would have to be a mandatory child of all three features, which is clearly a violation of the tree structure.

On the opposite side, it would be possible to define mandatory, optional, alternative and multiple features solely with restrictions. For example, the fact that features f and f' are mutually exclusive (that is, alternate features) could be expressed by assigning the restriction $\neg f'$ to f and the restriction $\neg f$ to f'.

By extending this approach, it would be possible to replace the different categories defined for FMFeatureDecomposition in Section 4.4. We did not follow this direction because it would be easy to generate such restrictions from a decomposition, but it would be hard to translate them back into decompositions without proper annotation.

4.6 Class FMFeatureRelation

As we have seen in Section 4.5, a FMFeatureRestriction is a boolean expression that restricts whether a feature may be included in a feature selection or not. In this section, we define FMFeatureRelations, which work differently in that they impose requirements instead of restrictions.

For example, the relation $F_1 \text{ requires } F_2$ states that the inclusion of feature F_1 in a feature selection *requires* that feature F_2 is also selected. Similarly, the relation $F_1 \text{ excludes } F_2$ states that if feature F_1 is part of a feature selection then it is *required* that feature F_2 is not present.

Relations are implemented by the class `FMFeatureRelation`. A `FMFeatureRelation` refers to a number of `FMFeature`s in the role `feature`; these are the target features of the relation.

Class	**FMFeatureRelation**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	Defines relations for FMFeatures, for example dependencies on other FMFeatures, or conflicts with other FMFeatures. A FMFeature can only be part of a FMFeatureSelectionSet if all its relations are fulfilled.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
feature	FMFeature	*	ref	The FMFeature that is targeted by this FMFeature Relation.
restriction	FMConditionByFeatures AndAttributes	0..1	aggr	If given, the condition shall evaluate to true, in order for the FMFeatureRelation to be active.

Table 4.6: FMFeatureRelation

[constr_3660] Multiplicity of `FMFeatureRelation.feature` ⌈For each `FMFeatureRelation` at least one reference in the role `feature` shall exist.⌋ *()*

[TPS_FMDT_00020] Structure of `FMFeatureRelation` ⌈A `FMFeatureRelation` R establishes a binary relation between two features:

1. The `FMFeature` f which aggregates R.
2. The `FMFeature` f' which R refers to in the role `feature`.

A `FMFeatureRelation` is always directed from f to f'.

If R refers to features $\{f_1, f_2, \ldots, f_n\}$ in the role `feature`, then R establishes n such binary relations.⌋ *(RS_FMDT_00008)*

The particular type of a relation is specified by its `category` attribute, which is covered in Section 4.6.1. We have defined a number of predefined relation types, which are listed in Section 4.6.3.

Obviously, a feature shall not establish a relation to itself.

[constr_5001] `FMFeatureRelation` shall not establish self-references ⌈A `FMFeatureRelation` that is aggregated by a `FMFeature` f shall not reference f in the role `feature`. In other words: self-references are not allowed.⌋ *()*

[constr_5001] helps to avoid conflicting relations such as "f conflicts f". Note that [constr_5001] cannot prevent *all* possible conflicts; for example a feature f might require a feature f' which in turn conflicts with f. Since the `category` of a `FMFeatureRelation` is designed to be extensible (see [TPS_FMDT_00023]), it is not feasible to formulate constraints that cover all possible conflicts.

4.6.1 Attribute category

Because a FMFeatureRelation is an Identifiable, it has a category attribute.

[TPS_FMDT_00021] category attribute of FMFeatureRelation ⌈The attribute category of a FMFeatureRelation specifies the kind of relation that is implemented here.⌋*(RS_FMDT_00008)*

Section 4.6.3 presents an overview of all predefined relations.

[TPS_FMDT_00023] Extensibility of category attribute of FMFeatureRelation ⌈The attribute category of FMFeatureRelation can be extended by proprietary relation types that go beyond those that are defined in this section.⌋*(RS_FMDT_00008)*

For example, a company may define proprietary relations that are only used in-house (or shared with selected customers). It is obviously no longer possible to safely exchange such a model with *everybody*, but it is a valid use case for a limited audience.

4.6.2 Identifying and documenting FMFeatureRelations

Because FMFeatureRelation is based on Identifiable, it may contain the optional attributes shortName, introduction, and desc.

[TPS_FMDT_00052] Identifying FMFeatureRelations ⌈The attribute shortName can be used to distinguish relations in case a FMFeature aggregates several FMFeatureRelations.⌋*(RS_FMDT_00004)*

[TPS_FMDT_00061] Documenting FMFeatureRelations ⌈The attributes introduction and desc may be used to provide a human readable description for a FMFeatureRelation.⌋*(RS_FMDT_00004)*

4.6.3 Predefined Relations

[TPS_FMDT_00019] Predefined values for the category of FMFeatureRelation ⌈In the following list, f is the feature that aggregates a FMFeatureRelation R in the role relation, and $f_1, f_2, \ldots, f_n$ are the features that R refers to in the role feature.

REQUIRES f shall only be part of a feature selection if $f_1, f_2, \ldots, f_n$ are also part of this feature selection.

EXCLUDES If f is part of a feature selection, then $f_1, f_2, \ldots, f_n$ shall not be part of this feature selection.

RECOMMENDED_FOR If one or more of the referenced features are selected then it is recommended to also include this one.

DISCOURAGED_FOR Opposite of RECOMMENDED_FOR: it is not recommended to include this feature if one or more of the referenced features are selected.

IMPACTS Selecting this feature has impact on all of the referenced features. "Impacted by" means that if one or more of the referenced features are selected then this feature has impact on the selected referenced features.

FUNCTIONAL_DEPENDENT There is a functional dependency between this feature and the referenced features.

For the following relation, assume that FMFeatures $f'_1, f'_2, \ldots, f'_m$ is a set of features each of which aggregates a FMFeatureRelation R_i in the role relation, and $f_1, f_2, \ldots, f_n$ are the common[3] features that all R_i refer to in the role feature.

PROVIDES If $f_1, f_2, \ldots, f_n$ are part of a feature selection, then at least one of $f'_1, f'_2, \ldots, f'_m$ shall be part of this feature selection.

The details for RECOMMENDED_FOR, DISCOURAGED_FOR, IMPACTS and FUNCTIONAL_DEPENDENT shall be given in attributes introduction and desc because they cannot be formalized. For tools, this means that these relations give hints to the user making the configuration. In contrast, a corresponding restriction can be derived automatically for relations REQUIRES and EXCLUDES.

⌋(*RS_FMDT_00008*)

[TPS_FMDT_00044] Semantics of FMFeatureRelation ⌈Let S be a feature selection and f be a FMFeature with a FMFeatureRelation R that references FMFeatures $f_1, f_2, \ldots, f_n$ in the role feature. Then R defines the following conditions:

category of R is REQUIRES

$$\forall i \in \{1, \ldots, n\} : f \in S \Rightarrow f_i \in S$$

category of R is EXCLUDES

$$\forall i \in \{1, \ldots, n\} : f \in S \Rightarrow f_i \notin S$$

Next, S be a feature selection and $f'_1, f'_2, \ldots, f'_m$ be FMFeatures, each of which aggregates a FMFeatureRelation R_i that references a set of features FMFeatures F_i in the role feature. Assume that all R_i have the same category, and let $\{f_1, f_2, \ldots, f_n\} = F_1 \cap F_2 \cap \ldots \cap F_m$ be the common features of all relations R_i. Then R defines the following condition:

category of R_i is PROVIDES

$$\forall i \in \{1, \ldots, n\} : f_i \in S \Rightarrow \exists 1 \leq j \leq m : f'_j \in S$$

All other relations do not define formal relations. Instead, the attributes introduction and desc (which exist because FMFeatureRelation is based on Identifiable) may provide a human-readable description of the meaning of the restriction.

⌋(*RS_FMDT_00008*)

[3] The individual R_i may to refer to additional FMFeatures, but here we are only interested in the common subset.

Note that a FMFeatureRelation just *defines* a condition and does not demand that this condition is actually fulfilled. This is because a feature selection might be incomplete. Only in a *valid feature selection* (see [TPS_FMDT_00030]) the conditions have to be obeyed.

4.7 Hierarchy, Restrictions and Relations

In this chapter, we have defined three different ways to introduce relationships between features: the hierarchy (Section 4.4), restrictions (Section 4.5) and relations (Section 4.6):

1. The *hierarchy* (FMFeatureDecomposition) only affects features of the same category that have the same parent[4] in the feature tree. That is, *alternative* features only depend on their parent and on siblings that are also alternative features, *multiple* features only depend on their parent and on siblings that are also multiple features, and *optional* and *mandatory* features only depend on their parent.

2. Features with *restrictions* (FMFeatureRestriction) depend on other features in the same feature model or even in another feature model. Unlike before, the relative position within the hierarchy does not play a role here.

   This is a more powerful approach than hierarchical dependencies, but restrictions need to be handled with more care than those defined by hierarchy. For example, it is easy to introduce circular dependencies or contradictions with restrictions.

3. *Relations* among features (FMFeatureRelation) may also introduce dependencies between features regardless of their position in the feature tree, but their scope is more limited.

   However, unlike in a restriction, where a feature depends on other features, a relation may influence other features. If feature A *requires* feature B, then a feature selection which includes A also has to include B.

[4] More precisely, FMFeature which are referenced from the same FMFeatureDecomposition in the role feature.

5 Feature Selection

A feature model does not describe a single product, but a set of products with common characteristics – a *product line*. An individual product is described by a specific combination of features. To be valid, such a combination of features needs to adhere to the various constrains defined in the feature model: hierarchical structure (Section 4.4), restrictions (Section 4.5), and relations (Section 4.6).

[TPS_FMDT_00060] Purpose of `FMFeatureSelectionSet` ⌈In AUTOSAR, a set of features that describes a product is implemented by the class `FMFeatureSelectionSet`.⌋ *(RS_FMDT_00003)*

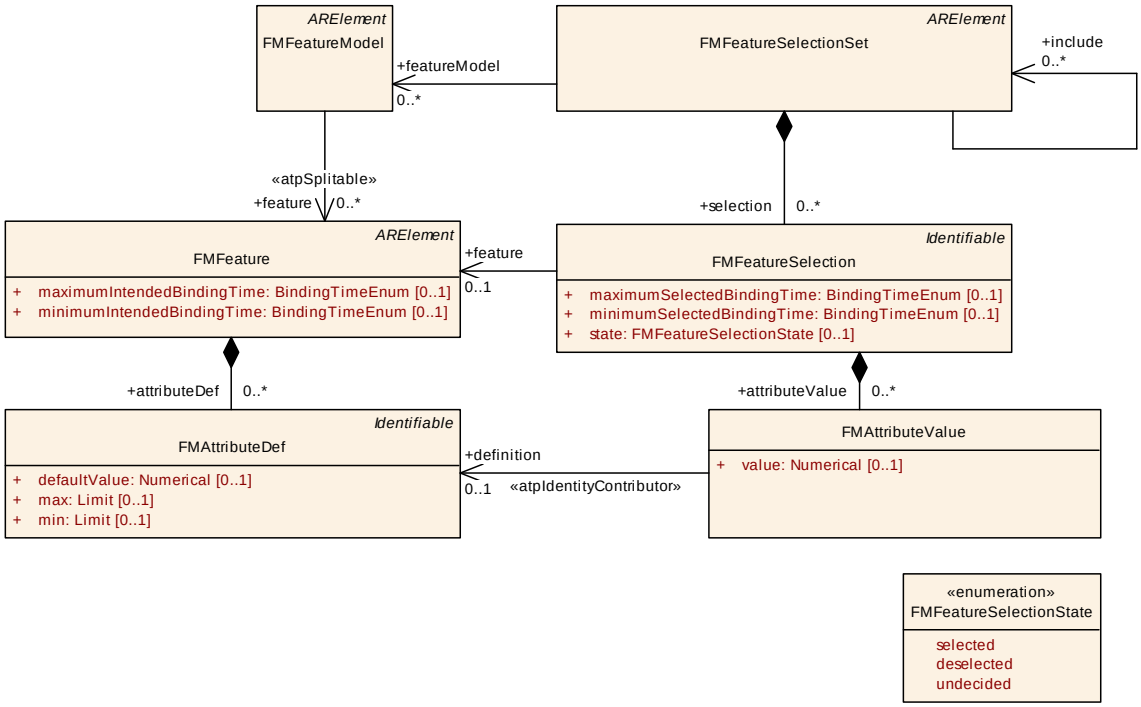

Figure 5.1: Class `FMFeatureSelectionSet`

5.1 Example

Table 5.1 shows an example for a feature selection.

Feature	Sports Edition	Family Edition
Engine	+	+
Gasoline Engine	+	−
Diesel Engine	−	+
Engine Controller	+	+
Gasoline Engine Controller	+	−
Diesel Engine Controller	−	+
Doors	+	+
Two Doors	+	+
Four Doors	−	+
Convertible	+	−
Sunroof	−	+
Electric window lift	+	+
Halogen lights	+	+

Table 5.1: Sample Feature Selection

We are re-using the example feature model 1.1 from Section 1.3 here. In our example, two feature selections are defined: *Sports Edition* and *Family Edition*, which correspond to two different car models. Some features, for example the halogen lights, are available in both models, while others are different: then *Sports Edition* uses a gasoline engine, while the *Family Edition* uses a diesel engine.

So, in its basic form, a feature selection is simply a list of features that are included in a variant[1], as indicated by the plus sign in example 5.1.

In our specification, we also allow variants to inherit from other variants. For example, all feature selections that are specific for a particular country (the famous “wheel on left/right side” distinction) may be contained in a separate feature model. A car model that is destined for a particular country can then simply include the country specific feature model.

5.2 Class `FMFeatureSelection`

A `FMFeatureSelection` represents a single `FMFeature`. The `FMFeatureSelection` has three attributes, `state`, `minimumSelectedBindingTime` and `maximumSelectedBindingTime`. The attribute `state` defines whether the feature is actually selected or not, or whether this is not yet decided. The attributes `minimumSelectedBindingTime` and `maximumSelectedBindingTime` define at which binding time the selection is supposed to happen.

[1] In our example, there are only variants for models of complete cars. This is of course an oversimplification; in the real world, variants are much more fine granular. For example, there could be country specific variants of *Sports Edition* or *Family Edition*.

Class	**FMFeatureSelection**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	A FMFeatureSelection represents the state of a particular FMFeature within a FMFeatureSelectionSet.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
attributeValue	FMAttributeValue	*	aggr	This defines a value for the attribute that is referred to in the role definition. Note that a FMFeatureSelection cannot include two FMAttributeValues that refer to the same FMAttributeDef in the role definition. **Tags:**xml.sequenceOffset=50
feature	FMFeature	0..1	ref	The FMFeature whose state is defined by this FMFeature Selection. **Tags:**xml.sequenceOffset=10
maximum SelectedBinding Time	BindingTimeEnum	0..1	attr	Defines an upper bound for the binding time of the variation points that are associated with the FMFeature, and refines its maximumIntendedBindingTime. This attribute is meant as a hint for the development process. **Tags:**xml.sequenceOffset=40
minimum SelectedBinding Time	BindingTimeEnum	0..1	attr	Defines a lower bound for the binding time of the variation points that are associated with the FMFeature, and refines its minimumIntendedBindingTime. This attribute is meant as a hint for the development process. **Tags:**xml.sequenceOffset=30
state	FMFeatureSelection State	0..1	attr	Defines how the FMFeature that is described by this FMFeatureSelection contributes to the FMFeature SelectionSet. A FMFeature may have the state selected, deselected or undecided. **Tags:**xml.sequenceOffset=20

Table 5.2: FMFeatureSelection

[constr_3661] Multiplicity of `FMFeatureSelection.feature` ⌈For each `FMFeatureSelection` the reference in the role `feature` shall exist.⌋*()*

[constr_3662] Multiplicity of `FMFeatureSelection.state` ⌈For each `FMFeatureSelection` the attribute `state` shall exist.⌋*()*

5.2.1 Reference `feature`

The reference `feature` points to the feature that is described by this `FMFeatureSelection`.

5.2.2 Attribute `state`

`FMFeatureSelection` has an attribute `state` that defines how the feature referred to by `feature` contributes to the selection.

Enumeration	**FMFeatureSelectionState**
Package	M2::AUTOSARTemplates::FeatureModelTemplate
Note	Defines how a particular FMFeature contributes to a FMFSelectionSet.
Literal	***Description***
deselected	The feature is excluded from the selection. **Tags:**atp.EnumerationLiteralIndex=0
selected	The feature is included in the selection. **Tags:**atp.EnumerationLiteralIndex=1
undecided	It is not yet decided whether the feature shall be included into or excluded from the selection. **Tags:**atp.EnumerationLiteralIndex=2

Table 5.3: FMFeatureSelectionState

The value `undecided` needs further explanation. In a `FMFeatureSelectionSet` F that is not included by another `FMFeatureSelectionSet`, the value `undecided` is not useful – in this case, a `FMFeature` should either have the `state` `selected` or `deselected` (or the `FMFeatureSelection` should be entirely missing).

However, if there is a `FMFeatureSelectionSet` F' that includes F, then it may be useful to set the value of `state` of a particular feature `FMFeature` f in F', and not in F. This cannot be done if f already has a `state` in F that is it is either `selected` or `deselected`. Hence, there is the need for a third value for `state` that can be overridden: `undecided`. For a more detailed explanation, wee Section 5.4.

In example 5.1, '$+$' corresponds to the state `selected`, and '$-$' corresponds to the state `deselected`. There are no `undecided` features because the example has deliberately been kept simple and does not use feature selections that include other feature selections.

5.2.3 `FMAttributeValue`

Each `FMFeatureSelection` aggregates a `FMAttributeValue` in the role `attributeValue`. This defines the value for a particular attribute of a feature in the context of this `FMFeatureSelection`.

Class	**FMAttributeValue**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	This defines a value for the attribute that is referred to in the role definition.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
definition	FMAttributeDef	0..1	ref	This refers to the definition of this attribute. **Stereotypes:** atpIdentityContributor
value	Numerical	0..1	attr	This represents the value of this attribute.

Table 5.4: FMAttributeValue

[constr_3663] Multiplicity of `FMAttributeValue.definition` ⌈For each `FMAttributeValue` the reference in the role `definition` shall exist.⌋*()*

[constr_3664] Multiplicity of `FMAttributeValue.value` ⌈For each `FMAttributeValue` the attribute `value` shall exist.⌋*()*

[TPS_FMDT_00053] Semantics of `FMAttributeValue` ⌈A `FMAttributeValue` defines a value for the `FMAttributeDef` that is referenced in the role `definition`. The particular value is stored in the attribute `value`.⌋*(RS_FMDT_00009)*

[constr_5027] Semantics of attributes `max` and `min` of `FMAttributeDef` in class `FMAttributeValue` ⌈Let v be the attribute `value` of an `FMAttributeValue` V that refers to `FMAttributeDef` D in the role `definition`. Furthermore, let min and max be the values of the attributes `min` and `max` of D.

The following condition shall hold true:

$$min \leq v \leq max$$

⌋*()*

Obviously, we do not want two `FMAttributeValue`s that refer to the same `FMAttributeDef`. Otherwise, it would not be clear which value to choose.

[constr_5028] Only one `FMAttributeValue` per `FMAttributeDef` ⌈Let S be a `FMFeatureSelectionSet` whose `FMFeatureSelection`s aggregate `FMAttributeValue`s $\{v_1, v_2, \ldots, v_n\}$ in the role `attributeValue`. For each v_i, let f_i be the `FMFeature` to which v_i refers to in the role `attributeDef`. Then the following condition shall hold:

$$\forall i \in \{1, \ldots, n\} : i \neq j \Rightarrow f_i \neq f_j$$

⌋*()*

5.2.3.1 Example

A feature may define an attribute named “pc” that specifies the power consumption for this feature, and whose values need to lie between 0 and 1000 milliwatt. In this case, the feature defines an `FMAttributeDef` where attribute `min` has the value 0, and `max` has the value 1000.

Furthermore, assume that `FMFeature` f has child features f_1, f_2, f_3, and f_4 that are all optional. All these features define an attribute named “pc”. Then f could add a `FMFeatureRestriction` to make sure that the power consumption of its child features does not exceed the power consumption allocated for f:

$$f_1.\mathrm{pc} + f_2.\mathrm{pc} + f_3.\mathrm{pc} + f_4.\mathrm{pc} < f.\mathrm{pc}$$

This can be useful if not every combination of f_1, f_2, f_3, and f_4 adheres to the allocated power consumption for f.

Furthermore, assume that the allowed power consumption of f depends on the car type. In this case, the `FMFeatureSelection` that refers f in the role `feature` may define a `FMAttributeValue` that overrides the default value given in f's `FMAttributeDef`.

5.2.4 Selected Binding Time

[TPS_FMDT_00055] Semantics of `minimumSelectedBindingTime` and `maximumSelectedBindingTime` ⌈These two attributes refine the attributes `minimumIntendedBindingTime` and `minimumIntendedBindingTime` that are defined at the `FMFeature` to which the `FMFeatureSelection` refers to in the role `feature`.⌋ *(RS_FMDT_00002, RS_FMDT_00016)*

[TPS_FMDT_00056] `minimumSelectedBindingTime` and `maximumSelectedBindingTime` are only hints ⌈The attributes `minimumSelectedBindingTime` and `maximumSelectedBindingTime` are only meant as hints for the development process to guide the selection of correct variability implementation.⌋ *(RS_FMDT_00002)*

5.3 Class `FMFeatureSelectionSet`

A `FMFeatureSelectionSet` aggregates an arbitrary number of `FMFeatureSelection` elements in the role of `selection`. Each `FMFeatureSelection` corresponds to a particular feature in a feature model, and states whether this feature is included into the selection or not.

Class	**FMFeatureSelectionSet**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	A FMFeatureSelectionSet is a set of FMFeatures that describes a specific product. **Tags:**atp.recommendedPackage=FMFeatureModelSelectionSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
featureModel	FMFeatureModel	*	ref	All FMFeatures in this FMFeatureSelectionSet shall be part of the referenced FMFeatureModel.
include	FMFeatureSelectionSet	*	ref	Each FMFeatureSelectionSet may include one or more FMFeatureSelectionSets. This establishes a hierarchy among FMFeatureSelectionSets. See constr_5003 and constr_5025 for details.
selection	FMFeatureSelection	*	aggr	The set of FMFeatureSelections of this FMFeature SelectionSet.

Table 5.5: FMFeatureSelectionSet

5.3.1 Terminology and constraints

FMFeatureSelectionSet aggregates its FMFeatureSelections, so it is not possible that a particular FMFeatureSelection is contained twice in a FMFeatureSelectionSet. However, two or more FMFeatureSelections could refer to the same FMFeature in the role feature, which could introduce ambiguities if the state of the FMFeatureSelections is different. Hence, we do not allow this.

[constr_5018] FMFeatureSelectionSet shall not include the same feature twice ⌈Let $\{s_1, s_2, \ldots, s_n\}$ be the set of FMFeatureSelection elements that are aggregated by a FMFeatureSelectionSet in the role selection. Furthermore, for each s_i, let f_i be the FMFeature that is referred to in the role feature. Then the following condition shall hold true:

$$\forall i, j \in \{1, 2, \ldots, n\} : i \neq j \Rightarrow f_i \neq f_j$$

⌋()

Constraint [constr_5018] makes sure that a FMFeatureSelectionSet assigns a unique state to each FMFeature in its associated FMFeatureModel.

[TPS_FMDT_00009] Definition of *Feature Set* of a FMFeatureSelectionSet ⌈Let S be a FMFeatureSelectionSet and $\{s_1, s_2, \ldots, s_n\}$ be the set of FMFeatureSelections aggregated by S in the role selection.

Then the *feature set* of S is the set of FMFeatures $\{f_1, f_2, \ldots, f_n\}$ where s_i refers to f_i in the role feature.⌋*(RS_FMDT_00003)*

[constr_5018] makes sure that if a FMFeatureSelectionSet aggregates n FMFeatureSelections, then its *feature set* also has the size n. However, a FMFeatureSelectionSet does not need to enumerate *all* FMFeatures of the associated FMFeatureModel. Nevertheless, all FMFeatures need to come from the same FMFeatureModel, as outlined in [constr_5023]:

[constr_5023] FMFeatureSelectionSet may only refer to FMFeatures from the associated FMFeatureModel ⌈Let S be a FMFeatureSelectionSet, and $\{f_1, f_2, \ldots, f_n\}$ be its *feature set* ([TPS_FMDT_00009]). Furthermore, let $\{g_1, g_2, \ldots, g_m\}$ be the combined *feature sets* of the FMFeatureModels to which S refers to in the role featureModel.

Then the following condition shall hold: $\{f_1, f_2, \ldots, f_n\} \subseteq \{g_1, g_2, \ldots, g_m\}$.⌋()

Note that if a FMFeature f is missing from a FMFeatureSelectionSet S, its state is not automatically equivalent to deselected. This would only be the case if S does not include another FMFeatureSelectionSet and is not included in another FMFeatureSelectionSet (see also 5.4).

5.3.2 Relation include

A FMFeatureSelectionSet may refer to other FMFeatureSelectionSets in the role include. If FMFeatureSelectionSet A includes FMFeatureSelectionSet B, then the total features selected by A is the sum of the features selected by A and the features selected by B.

[constr_5024] FMFeatureSelectionSet shall not include itself ⌈Let S be a FMFeatureSelectionSet and let S' be the FMFeatureSelectionSet to which S refers to in the role include.

Then the following condition shall hold: $S \neq S'$.⌋*()*

Next, we define a graph structure that describes the include relations among FMFeatureSelectionSets:

[TPS_FMDT_00032] *Inclusion graph* for FMFeatureSelectionSets ⌈Let $\{S_1, S_2, \ldots, S_n\}$ be the set of all FMFeatureSelectionSets in an AUTOSAR model. Then the *inclusion graph* for all FMFeatureSelectionSets is a graph $G = (V, E)$ where

$$V = \{S_1, S_2, \ldots, S_n\}$$

$$E = \{(S_i, S_j) \mid S_i \text{ refers to } S_j \text{ in the role include}\}$$

⌋*(RS_FMDT_00003)*

Obviously, the inclusion graph for an AUTOSAR model is allowed to contain isolated nodes – FMFeatureSelectionSets that stand on their own and do not include other FMFeatureSelectionSet or are included elsewhere.

[constr_5024] can also be described in terms of the *inclusion graph*: the *inclusion graph* does not allow self loops. With the next constraint, we generalize this constraint and disallow cycles in the include relations:

[constr_5002] FMFeatureSelectionSet shall not have cycles in the include relation ⌈Let S be a FMFeatureSelectionSet and let G be the *inclusion graph* for all FMFeatureSelectionSets as defined in [TPS_FMDT_00032]. There shall be no cycles in the inclusion graph.⌋*()*

5.4 state and include

Consider the following situation. FMFeatureSelectionSets S, S_1 and S_2 include FMFeatureSelections that refer to the same FMFeature f. Let s, s_1 and s_2 be the value of the attribute state of the FMFeatureSelection that refers to f in S, S_1 and S_2, respectively.

Two questions arise from that:

1. If S includes S_1, which values may s assume?
2. If S includes S_1 and S_2, which combination of values for s_1 and s_2 are allowed and which values may s assume?

In case 1, s should never override s_1. That is, if s_1 is already `selected`, then s cannot be `deselected`, but it may be `undecided`. Vice versa, if s_1 is already `deselected`, then s cannot be `selected`, but it may be `undecided`. Finally, if s_1 is `undecided`, then s may assume any value.

[constr_5003] `FMFeatureSelectionSet` shall not overwrite the state of included features ⌈Let S be a `FMFeatureSelectionSet` that aggregates a `FMFeatureSelection` that has the `state` s and which refers to a `FMFeature` f in the role `feature`. Furthermore, let S_1 be a `FMFeatureSelectionSet` that aggregates a `FMFeatureSelection` that has the `state` s_1 and refers *to the same* `FMFeature` f in the role `feature`. Finally assume that S refers to S_1 in the role `include`.

Then the following conditions shall hold:

1. If the value of the attribute `state` of s_1 is `undecided`, then the value of the attribute `state` of s may be one of `selected`, `deselected`, and `undecided`.
2. If the value of the attribute `state` of s_1 is `selected` or `deselected`, then the value of the attribute `state` of s shall be the same as the attribute `state` in s_1, or `undecided`.
3. Any other constellation is considered an error.

⌋()

	s (`state` in S)	s_1 (`state` in S_1)
valid	selected	selected
invalid	*selected*	*deselected*
valid	selected	undecided
invalid	*deselected*	*selected*
valid	deselected	deselected
valid	deselected	undecided
valid	undecided	selected
valid	undecided	deselected
valid	undecided	undecided

Table 5.6: Summary: `FMFeatureSelectionSet` S includes S_1.

The behavior is summarized in Table 5.6. Some combinations are labeled as **`invalid`**; these are the cases where the `state` of a feature that is already `selected` or `deselected` would be overwritten with a different value.

In case 2, the difference is that there is not just a s_1, but also a s_2. So, we need to make sure that s_1 and s_2 do not make contradictory statements about f. That is, it should not happen that s_1 is `selected` and s_2 is deselected, or vice versa. Again, an `undecided` in s_1 or s_2 is uncritical.

[constr_5025] `FMFeatureSelectionSet` shall not overwrite the state of included features ⌈Let S be a `FMFeatureSelectionSet` that aggregates a `FMFeatureSelection` that has the `state` s and which refers to a `FMFeature` f in the role `feature`. Furthermore, let S_1 (S_2) be a `FMFeatureSelectionSet` that aggregates a `FMFeatureSelection` that has the `state` s_1 (s_2) and refers *to the same* *`FMFeature`* f in the role `feature`. Finally assume that S refers to S_1 *and* S_2 in the role `include`.

Then the following conditions shall hold:

1. If the values of the attributes `state` of s_1 and s_2 are both `undecided`, then the value of the attribute `state` of s may be `selected`, `deselected` or `undecided`.
2. If the value of the attribute `state` of s_1 is `undecided` and the value of the attribute `state` of s_2 is `selected` or `deselected`, then the value of the attribute `state` of s shall be the same as the attribute `state` in s_2, or `undecided`.
3. If the value of the attribute `state` of s_2 is `undecided` and the value of the attribute `state` of s_1 is `selected` or `deselected`, then the value of the attribute `state` of s shall be the same as the attribute `state` in s_1, or `undecided`.
4. If the values of the attributes `state` of s_1 and s_2 are both either `selected` or `deselected`, then the value of the attribute `state` of s shall be the same as in attribute s_1, or `undecided`.
5. Any other constellation is considered an error.

⌋()

This behavior is summarized in Table 5.7.

	s (`state` in S)	s_1 (`state` in S_1)	s_2 (`state` in S_2)
valid	selected	selected	selected
invalid	selected	selected	deselected
valid	selected	selected	undecided
invalid	selected	deselected	selected
invalid	selected	deselected	deselected
invalid	selected	deselected	undecided
valid	selected	undecided	selected
invalid	selected	undecided	deselected
valid	selected	undecided	undecided
invalid	deselected	selected	selected
invalid	deselected	selected	deselected
invalid	deselected	selected	undecided
invalid	deselected	deselected	selected
valid	deselected	deselected	deselected
valid	deselected	deselected	undecided
invalid	deselected	undecided	selected
valid	deselected	undecided	deselected
valid	deselected	undecided	undecided
valid	undecided	selected	selected
invalid	undecided	selected	deselected
valid	undecided	selected	undecided

invalid	undecided	deselected	selected
valid	undecided	deselected	deselected
valid	undecided	deselected	undecided
valid	undecided	undecided	selected
valid	undecided	undecided	deselected
valid	undecided	undecided	undecided

Table 5.7: Summary: `FMFeatureSelectionSet` S includes S_1 and S_2.

5.5 Valid Feature Selection

[TPS_FMDT_00030] Definition of *Valid Feature Selection* ⌈Let S be a `FMFeatureSelectionSet` and F be the *feature set* of S. S is a *valid feature selection* if all the following constraints are obeyed:

- [TPS_FMDT_00046] (Semantics of `FMFeatureDecomposition`)
- [TPS_FMDT_00045] (Semantics of `FMFeatureRestriction`)
- [TPS_FMDT_00044] (Semantics of `FMFeatureRelation`)

⌋*(RS_FMDT_00003, RS_FMDT_00005, RS_FMDT_00008)*

6 Feature Map

In AUTOSAR variant handling, variation points are controlled by system constants. Each variation point contains a boolean expression[1] which determines whether this variation point is “on” or “off”. The AUTOSAR formula language allows references to SwSystemconsts as operands (see [TPS_GST_00001]). This is the same type of expressions that are used to model restrictions for FMFeatures, except that those are based on references to other FMFeatures in place of references to SwSystemconsts.

So, in order to associate features with variation points, we need a data structure that assigns values to SwSystemconsts based on which features are selected. This is implemented by the class FMFeatureMap.

6.1 Example

In example 1.1, we introduced an optional feature “Four Doors” which adds two more doors to the car model. Example 6.1 shows a small clipping of the XML representation for a car model which contains two SwComponentPrototypes named LeftDoorController and RightDoorController that are subject to variation.

Listing 6.1: Sample variation points `LeftDoorController` and `RightDoorController`

```
<SW-COMPONENT-PROTOTYPE>                                                    1
  <SHORT-NAME>LeftDoorController</SHORT-NAME>                               2
  <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">                          3
    DoorController                                                          4
  </TYPE-TREF>                                                              5
  <VARIATION-POINT>                                                         6
    <SHORT-LABEL>Left</SHORT-LABEL>                                         7
    <SW-SYSCOND BINDING-TIME="SYSTEM-DESIGN-TIME">                          8
      <SYSC-REF DEST="SW-SYSTEMCONST">HAS_LEFT_DOOR_CNTLR</SYSC-REF> == 1   9
    </SW-SYSCOND>                                                           10
  </VARIATION-POINT>                                                        11
</SW-COMPONENT-PROTOTYPE>                                                   12
<SW-COMPONENT-PROTOTYPE>                                                    13
  <SHORT-NAME>RightDoorController</SHORT-NAME>                              14
  <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">                          15
    DoorController                                                          16
  </TYPE-TREF>                                                              17
  <VARIATION-POINT>                                                         18
    <SHORT-LABEL>Right</SHORT-LABEL>                                        19
    <SW-SYSCOND BINDING-TIME="SYSTEM-DESIGN-TIME">                          20
      <SYSC-REF DEST="SW-SYSTEMCONST">HAS_RIGHT_DOOR_CNTLR</SYSC-REF> == 1  21
    </SW-SYSCOND>                                                           22
  </VARIATION-POINT>                                                        23
</SW-COMPONENT-PROTOTYPE>                                                   24
```

[1] We are simplifying here a bit; this is strictly true only for non-PostBuild variation points. PostBuild variation points do not use expressions, but compare the value of a system constant to a particular PostBuildVariantCondition.

The conditions for the variation points are in lines 9 and 21. In these conditions, we refer to system constants HAS_LEFT_DOOR_CNTLR and HAS_RIGHT_DOOR_CNTLR and check whether they have the value 1.

Assume we have a FMFeatureSelectionSet which contains a FMFeatureSelection that refers to the feature named “Four Doors” and has the state selected (see Section 5.2.2). Then we need to make sure that the value 1 gets assigned to *both* the system constants HAS_LEFT_DOOR_CNTLR and HAS_RIGHT_DOOR_CNTLR. In a pseudo programming language notion, this would look as follows:

```
if has_feature('Four Doors') == 1 then
  set_sysc('HAS_LEFT_DOOR_CNTLR', 1)
  set_sysc('HAS_RIGHT_DOOR_CNTLR', 1)
end
```

This shows that a feature can affect more than one system constant.

To extend our example further, lets assume that there is a constraint that prevents the controllers in this example to be used in non-european countries. (This could also be added as a restriction to the feature model, but such technical constraints are sometimes handled as part of the mapping.) Instead, an alternate controller is used in these countries. We need to extend the above pseudo code accordingly:

```
if has_feature('Four Doors') == 1 && has_feature('EuropeanCountry') == 1 then
  set_sysc('HAS_LEFT_DOOR_CNTLR', 0)
  set_sysc('HAS_RIGHT_DOOR_CNTLR', 0)
end
if has_feature('Four Doors') == 1 && has_feature('EuropeanCountry') == 0 then
  set_sysc('HAS_ALTERNATE_LEFT_DOOR_CNTLR', 1)
  set_sysc('HAS_ALTERNATE_RIGHT_DOOR_CNTLR', 1)
end
```

Now, we have *two* sets of assignments to system constants as well as complex conditions.

6.2 Overview

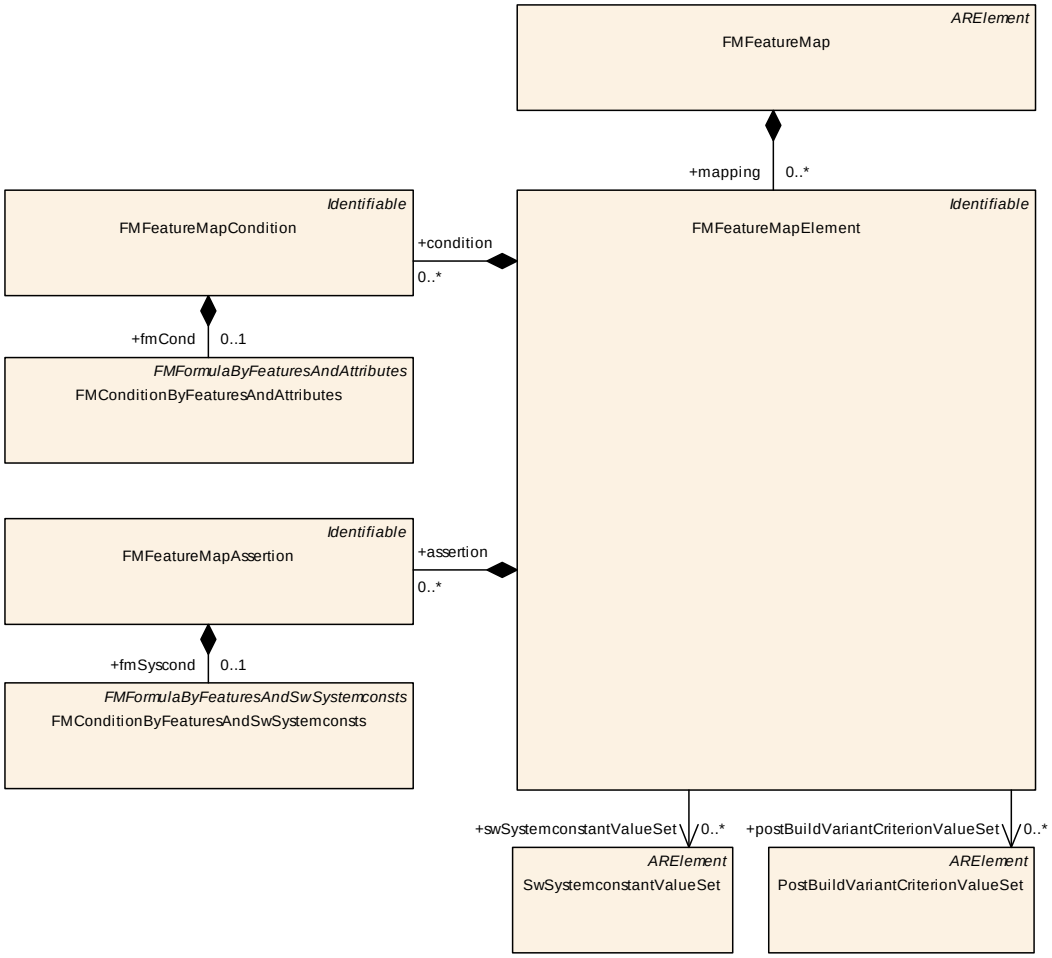

Figure 6.1: Class `FMFeatureMap`

A `FMFeatureMap` aggregates a number of `FMFeatureMapElement`s:

- In the simplest case, a `FMFeatureMapElement` chooses a value for a system constant if a certain feature is selected or deselected.
- In the general case, a `FMFeatureMapElement` chooses values for a set of system constants and postbuild variant criteria if a certain combination of features is selected.

We use the term “chooses” instead of “assigns” in the previous paragraph because an assignment would imply that a system constant behaves like a variable in a typical programming language that can be declared, perhaps initialized and later assigned a value.

This is not the case here. First, AUTOSAR does not have a concept akin to “assign a value later”. System constants can only be declared and initialized, but not changed afterwords. Second, feature models are optional, so all systems constants need to be declared and initialized in the non-optional part of the AUTOSAR model, which does not (and cannot) know about feature models.

6.3 Class FMFeatureMap

A FMFeatureMap aggregates a number of FMFeatureMapElements in the role mapping.

Class	FMFeatureMap			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	A FMFeatureMap associates FMFeatures with variation points in the AUTOSAR model. To do this, it defines value sets for system constants and postbuild variant criterions that shall be chosen whenever a certain combination of features (and system constants) is encountered. **Tags:**atp.recommendedPackage=FMFeatureMaps			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
mapping	FMFeatureMapElement	*	aggr	Set of mappings defined by this FMFeatureMap.

Table 6.1: FMFeatureMap

6.4 Class FMFeatureMapElement

Class	FMFeatureMapElement			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	Defines value sets for system constants and postbuild variant criterions that shall be chosen whenever a certain combination of features (and system constants) is encountered.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
assertion	FMFeatureMap Assertion	*	aggr	Defines a boolean expression based on features and system constants which needs to evaluate to true for this mapping to become active.
condition	FMFeatureMap Condition	*	aggr	Defines a condition which needs to be fulfilled for this mapping to become active.
postBuildVariant CriterionValue Set	PostBuildVariant CriterionValueSet	*	ref	Selects a set of values for postbuild variant criterions.
sw Systemconstant ValueSet	SwSystemconstant ValueSet	*	ref	Selects a set of values for system constants.

Table 6.2: FMFeatureMapElement

Each FMFeatureMapElement contains two kinds of assertions:

- A number of FMFeatureMapConditions in the role condition.

  A FMFeatureMapCondition aggregates a boolean expression of class FMFormulaByFeaturesAndAttributes in the role fmCond (see Section 7.2.1). This is the same kind of expression that is used by FMFeature to implement FMFeatureRestrictions. In fact, it serves a very similar purpose: the FMFeatureMapElement is only active if *at least one* of its FMFeatureMapConditions evaluates to *true*.

- A number of FMFeatureMapAssertions in the role assertion.

  An FMFeatureMapAssertion aggregates a boolean expression FMConditionByFeaturesAndSwSystemconsts in the role fmSyscond (see Section 7.2.4). The FMFeatureMapElement is only active if *all* its FMFeatureMapAssertions (more precisely, the formulas aggregated in the role fmSyscond) evaluate to *true*.

Both FMFeatureMapCondition and FMFeatureMapAssertion are Identifiables, which means that each of them has a shortName attribute that can be used to identify individual conditions and assertions, as well as desc and introduction for documentation purposes.

There are also two elements that choose values for system constants resp. values for postbuild variant criteria:

- A number of SwSystemconstantValueSet that are referenced in the role swSystemconstantValueSet.
- A number of PostBuildVariantCriterionValueSets that are referenced in the role postBuildVariantCriterionValueSet.

The rationale for using choosing values for more than one system constant or postbuild variant criterion per feature is that features are a more high-level concept than variation points. For example, a feature that switches between two different software components (such as a “basic” and a “comfort” variant) actually triggers several variation points: not just the software components change, but also their ports and their connectors. Unless all variation points depend on the same system constant, this means that we need to choose values for several system constants.

6.5 Relationship with PredefinedVariant

The classes SwSystemconstantValueSet and PostBuildVariantCriterionValueSet are originally part of the PredefinedVariant structure from variant handling ([1]). A PredefinedVariant represents a particular variant as a given combination of settings of variant selectors represented by SwSystemconstValue respectively PostBuildVariantCriterionValue ([TPS_GST_00280]).

A PredefinedVariant can be seen as a list of SwSystemconstantValueSets and PostBuildVariantCriterionValueSets. This is very similar to a mapping that has no conditions and no assertions.

Indeed, we could have used a reference to a PredefinedVariant instead of references to SwSystemconstantValueSets and PostBuildVariantCriterionValueSets. However, PredefinedVariants usually have a much coarser granularity than what is needed in a FMFeatureMapElement. So, instead of requiring to

adapt the granularity of PredefinedVariants, we refer to individual SwSystemconstantValueSets and PostBuildVariantCriterionValueSets. Usually, these will be a subset of the same PredefinedVariant.

A typical way to construct a FMFeatureMapElement is to look at the corresponding PredefinedVariant and then select those SwSystemconstantValueSets and PostBuildVariantCriterionValueSets that are relevant for the given mapping.

6.6 So, how does it work?

[TPS_FMDT_00037] Semantics of FMFeatureMapElement ⌈Let M be a FMFeatureMapElement. If the following expressions evaluate to *true*

1. At least one of the FMFeatureMapCondition elements that are referenced from M in the role condition
2. All FMFeatureMapAssertions that are referenced from M in the role assertion

then a processor shall use the SwSystemconstantValueSets which are referenced from M in the role swSystemconstantValueSet as well as the PostBuildVariantCriterionValueSets which are referenced from M in the role postBuildVariantCriterionValueSet to choose values for the associated SwSystemconsts respectively PostBuildVariantCriterions.⌋*(RS_FMDT_00010)*

Class	**FMFeatureMapCondition**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	Defines a condition which needs to be fulfilled for this mapping to become active. The condition is implemented as formula that is based on features and attributes and is defined by fmCond.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
fmCond	FMConditionByFeatures AndAttributes	0..1	aggr	The formula that implements the condition.

Table 6.3: FMFeatureMapCondition

Class	**FMFeatureMapAssertion**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	Defines a boolean expression which shall evaluate to true for this mapping to become active. The expression is a formula that is based on features and system constants, and is defined by fmSyscond.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
fmSyscond	FMConditionByFeatures AndSwSystemconsts	0..1	aggr	The formula that implements the assertion.

Table 6.4: FMFeatureMapAssertion

Class	**SwSystemconstantValueSet**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to specify a set of system constant values. **Tags:**atp.recommendedPackage=SwSystemconstantValueSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
sw Systemconstant Value	SwSystemconstValue	*	aggr	This is one particular value of a system constant.

Table 6.5: SwSystemconstantValueSet

Class	**PostBuildVariantCriterionValueSet**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to denote one set of postBuildVariantCriterionValues. **Tags:**atp.recommendedPackage=PostBuildVariantCriterionValueSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
postBuildVariant CriterionValue	PostBuildVariant CriterionValue	*	aggr	This is is one particular postbuild variant criterion/value pair being part of the PostBuildVariantSet.

Table 6.6: PostBuildVariantCriterionValueSet

6.7 Which variation points are affected by a particular `FMFeature`?

Because a `FMFeatureMap` does not directly refer to `VariationPoint` elements, it is not straightforward to see which variation points are affected by a particular feature.

First, we need to look at `SwSystemconstantValueSet` and `PostBuildVariantCriterionValueSet` to see which `SwSystemconst` and `PostBuildVariantCriterion` elements are actually affected by these. Figures 6.2 and 6.3 illustrate this.

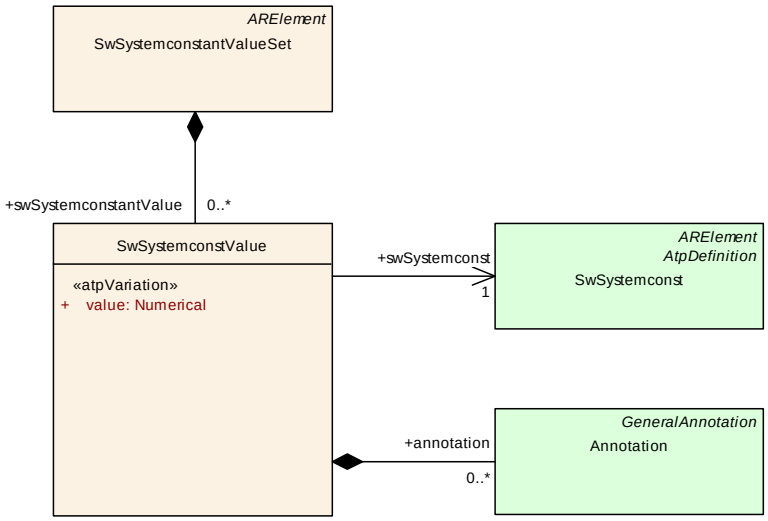

Figure 6.2: `SwSystemconstantValueSet` and `SwSystemconstValue`

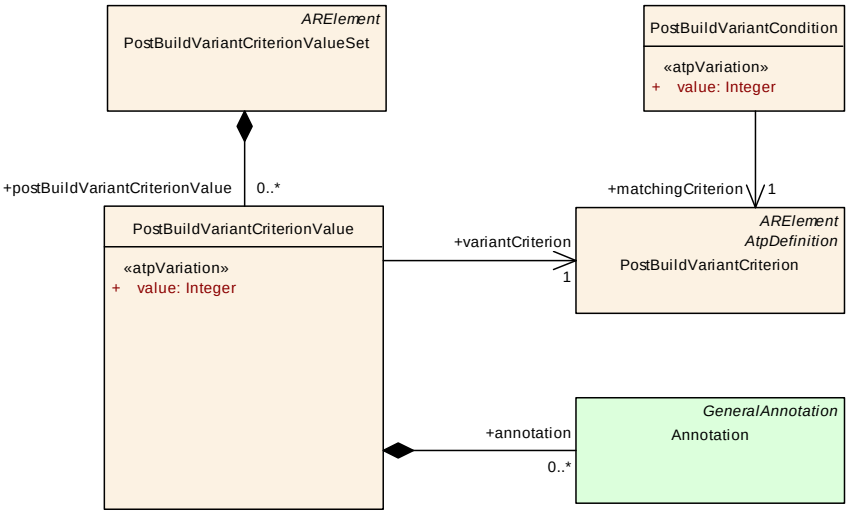

Figure 6.3: `PostBuildVariantCriterionValueSet` and `PostBuildVariantCriterionValue`

Next, we need to look at all variation points to see where those `SwSystemconst` and `PostBuildVariantCriterion` are referenced. Figure 6.4 shows the structure of a variation point. While the `PostBuildVariantCriterion` is directly visible in Figure 6.4, a `SwSystemconst` would be referenced from a `ConditionByFormula`.

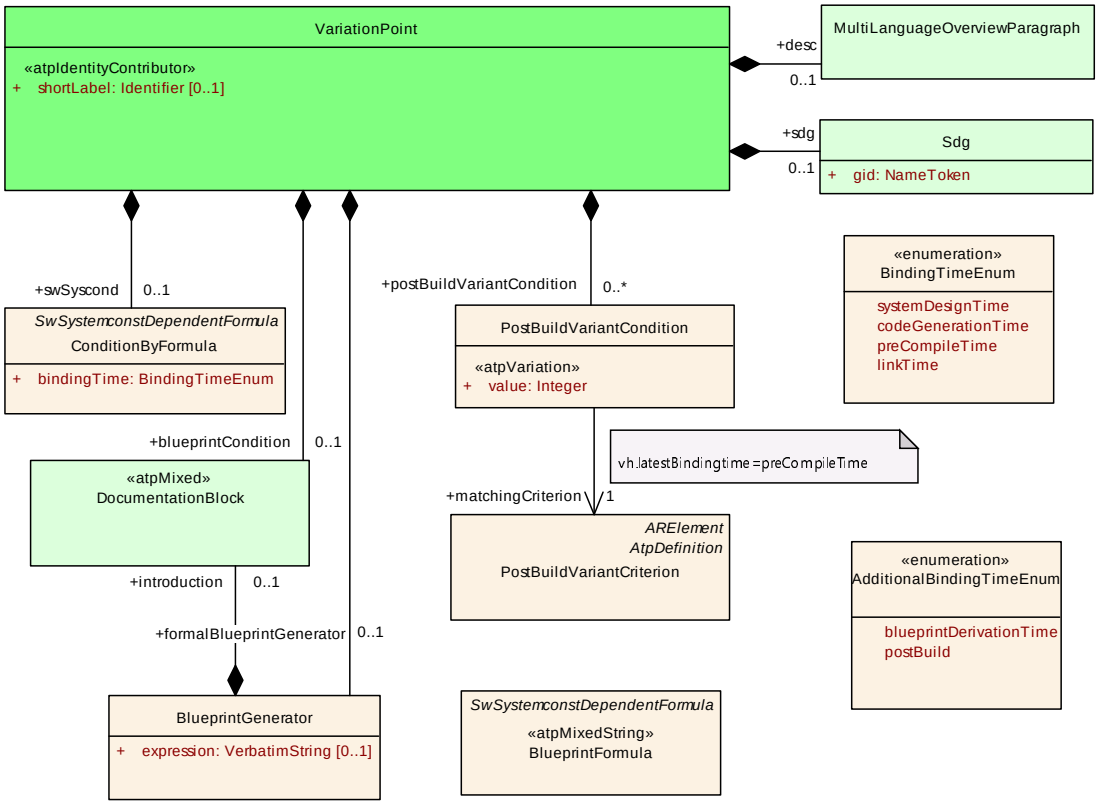

Figure 6.4: Variation Point

The algorithm to find out which variation points are affected by a single `FMFeatureMapElement` is outlined in [TPS_FMDT_00025].

[TPS_FMDT_00025] Set of *affected variation points* for a `FMFeatureMapElement`

⌈

1. Let e be a `FMFeatureMapElement`.
2. Let $S = \{s_1, s_2, \ldots, s_n\}$ the set of `SwSystemconstantValueSet` elements aggregated by e in the role `swSystemconstantValueSet`.
3. Each `SwSystemconstantValueSet` s_i aggregates a number of `SwSystemconstValue` elements in the role `swSystemconstantValue`, which in turn refer to a single `SwSystemconst` element in the role `swSystemconst`. Let SC_i be the set of these `SwSystemconst` elements for each s_i.
4. Each `SwSystemconst` in SC_i is used in the condition of one or more `VariationPoint`s. More precisely, the `SwSystemconst`s are referenced from the `ConditionByFormula` elements[2] that are aggregated by the `VariationPoint` in the role `swSyscond`. Let V_i be the set of these variation points for all elements in SC_i.

[2] `ConditionByFormula` elements are expressions which use `SwSystemconst` elements as variables.

5. Let $P = \{p_1, p_2, \ldots, p_m\}$ be the set of `PostBuildVariantCriterionValueSet` elements aggregated by e in the role `postBuildVariantCriterionValueSet`.
6. Each `PostBuildVariantCriterionValueSet` p_j aggregates a number of `PostBuildVariantCriterionValue` elements in the role `postBuildVariantCriterionValue`, which in turn refer to a `PostBuildVariantCriterion` in the role `variantCriterion`. Let PC_j be the set of `PostBuildVariantCriterion` elements.
7. Each `PostBuildVariantCriterion` element is used in a variation point. More precisely, a `VariationPoint` aggregates a `PostBuildVariantCondition` which in turn references a `PostBuildVariantCriterion`. Let V'_j be the set of these variation points for all elements in PC_j.

The *affected variation points* for the `FMFeatureMapElement` e is now defined as follows:

$$\text{affected variation points}(e) = \bigcup_i V_i \cup \bigcup_j V'_j$$

⌋*(RS_FMDT_00010)*

With [TPS_FMDT_00025] in place, we can now collect the affected variation points for a `FMFeature`.

[TPS_FMDT_00038] Definition of *Affected Variation Points* for a `FMFeature` ⌈

1. Let f be a `FMFeature`.
2. Let C be the set of `FMFeatureMapElement` elements that aggregate a `FMFeatureMapCondition` in the role `fmCond` whose aggregated `FMFormulaByFeaturesAndAttributes` element refers to f.
3. Let A be the set of `FMFeatureMapElement` elements that aggregate a `FMFeatureMapAssertion` in the role `fmSyscond` whose aggregated `FMConditionByFeaturesAndSwSystemconsts` element refers to f.

$$\text{affected variation points}(f) = \text{affected variation points}(C) \cup \text{affected variation points}(A)$$

⌋*(RS_FMDT_00010)*

7 Common Concepts

7.1 Special Data in Context of Feature Models

Usually, a feature model is maintained in an external system, and the AUTOSAR representation of a feature model is an export of that model. In order to maintain the relationship with the external model (or with other systems that might interact with the feature model), it is usually necessary to add application specific data, for example a custom identifier, to the feature model.

[TPS_FMDT_00033] Special data for feature models ⌈Several of the major classes in the feature model concept are based on the abstract class `ARElement`:

- `FMFeatureModel`
- `FMFeature`
- `FMFeatureSelectionSet`
- `FMFeatureMap`

The following classes are based on the abstract class `Identifiable`:

- `FMFeatureRestriction`
- `FMFeatureRelation`
- `FMAttributeDef`
- `FMFeatureMapCondition`
- `FMFeatureMapAssertion`

These classes aggregate `AdminData`in the role `adminData`, which aggregates a `Sdg` (special data group) in the role `sdg`. `Sdg` is a container that is designed to hold proprietary, application specific data that may be used by the application that exports the feature model into the AUTOSAR model to add its own data.⌋*(RS_FMDT_00001, RS_FMDT_00012, RS_FMDT_00013)*

Obviously, this also means that the data that is contained in the `Sdg` is not suited for exchange between arbitrary parties, but only between those who know how to interpret it.

7.2 Formulas that use Features

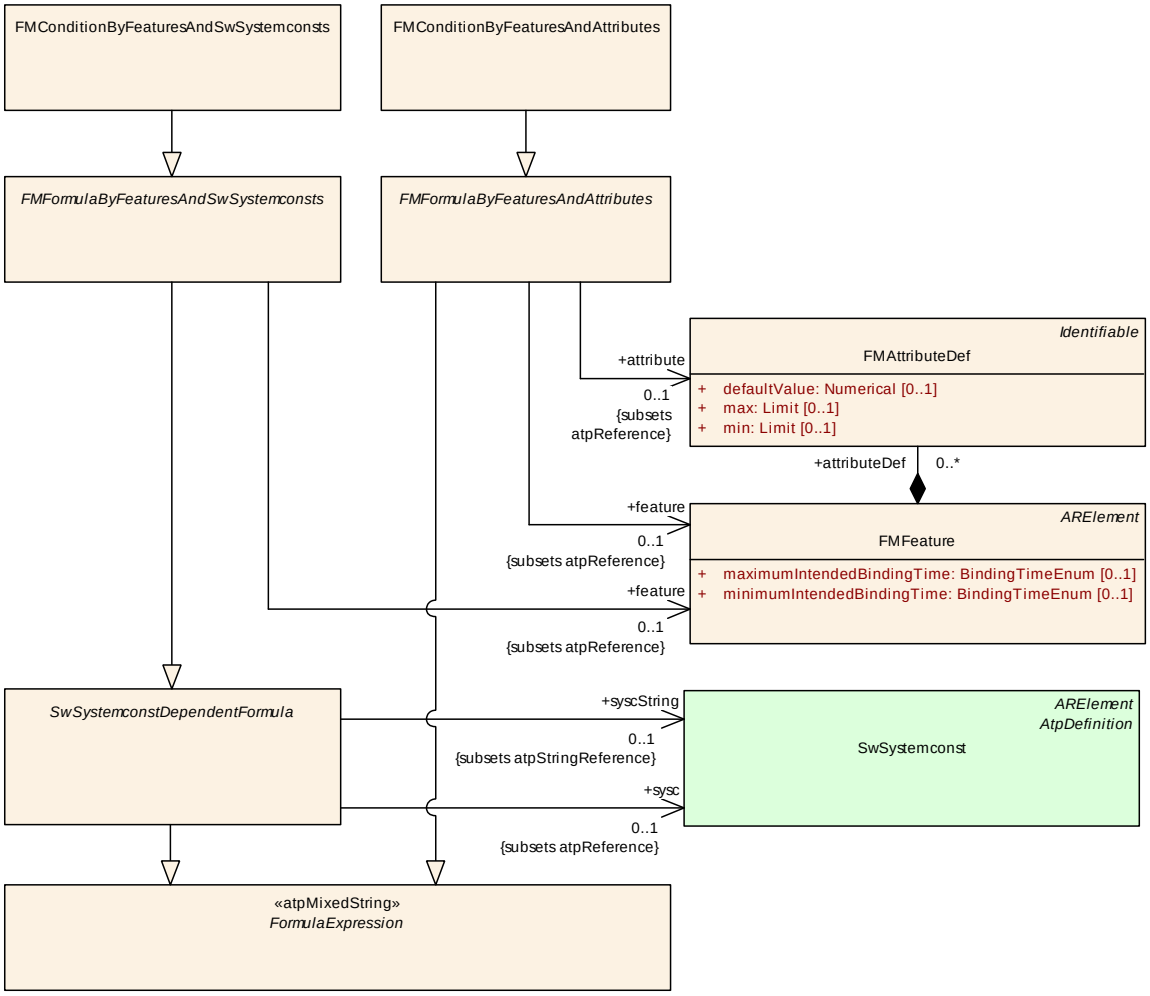

Figure 7.1: Formulas used in Feature Modeling

7.2.1 FMFormulaByFeaturesAndAttributes

Class	<<atpMixedString>> ***FMFormulaByFeaturesAndAttributes*** (abstract)			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	An expression that has the syntax of the AUTOSAR formula language but uses only references to features or feature attributes (not system constants) as operands.			
Base	*ARObject*, *FormulaExpression*			
Subclasses	FMConditionByFeaturesAndAttributes			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
attribute	FMAttributeDef	0..1	ref	An expression of type FMFormulaByFeaturesAndAttributes may refer to attributes of FMFeatures.

△

Class	<<atpMixedString>> ***FMFormulaByFeaturesAndAttributes*** (abstract)			
feature	FMFeature	0..1	ref	An expression of type FMFormulaByFeaturesAnd Attributes may refer to FMFeatures.

Table 7.1: FMFormulaByFeaturesAndAttributes

[constr_3665] Multiplicity of `FMFormulaByFeaturesAndAttributes.attribute` ⌈For each `FMFormulaByFeaturesAndAttributes` the reference in the role `attribute` shall exist.⌋*()*

[constr_3666] Multiplicity of `FMFormulaByFeaturesAndAttributes.feature` ⌈For each `FMFormulaByFeaturesAndAttributes` the reference in the role `feature` shall exist.⌋*()*

The class `FMFormulaByFeaturesAndAttributes` defines expressions that employ the same structure as the standard AUTOSAR formula language (see [1]), but use features and feature attributes instead of system constants.

This is expressed by the abstract class `FMFormulaByFeaturesAndAttributes`, which is based on `FormulaExpression` but restricts formulas so that they can only refer to `FMFeature`s and `FMAttributeDef`s, but not to `SwSystemconst`s.

[constr_5011] `FMFormulaByFeaturesAndAttributes` can refer to `FMFeature`s and `FMAttributeDef`s, but not to system constants ⌈A formula of class `FMFormulaByFeaturesAndAttributes` is an expression that can use `FMFeature`s and `FMAttributeDef`s, but is not allowed to use `SwSystemconst`s.⌋*()*

System constants are not allowed in this class of formulas because system constants are considered part of the implementation, whereas features (and feature attributes) abstract from the implementation.

7.2.2 `FMConditionByFeaturesAndAttributes`

Class	<<atpMixedString>> **FMConditionByFeaturesAndAttributes**			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	A boolean expression that has the syntax of the AUTOSAR formula language but uses only references to features or feature attributes (not system constants) as operands.			
Base	*ARObject*, *FMFormulaByFeaturesAndAttributes*, *FormulaExpression*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 7.2: FMConditionByFeaturesAndAttributes

[TPS_FMDT_00049] The result of `FMConditionByFeaturesAndAttributes` is interpreted as a boolean value. ⌈The result of a formula of class `FMConditionByFeaturesAndAttributes` shall be interpreted as a boolean value where 0 shall be interpreted as *false* and any value different from 0 shall be interpreted as *true*. This

is the same approach as used by ConditionByFormula.⌋(*RS_FMDT_00008, RS_FMDT_00010*)

An element of class FMConditionByFeaturesAndAttributes is aggregated by class FMFeatureRestriction in the role restriction and by class FMFeatureMapCondition in the role fmCond.

7.2.3 FMFormulaByFeaturesAndSwSystemconsts

Class	<<atpMixedString>> ***FMFormulaByFeaturesAndSwSystemconsts*** (abstract)			
Package	M2::AUTOSARTemplates::FeatureModelTemplate			
Note	An expression that has the syntax of the AUTOSAR formula language and may use references to features or system constants as operands.			
Base	*ARObject*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Subclasses	FMConditionByFeaturesAndSwSystemconsts			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
feature	FMFeature	0..1	ref	An expression of type FMFormulaByFeaturesAndSw Systemconsts may refer to FMFeatures.

Table 7.3: FMFormulaByFeaturesAndSwSystemconsts

[constr_3667] Multiplicity of FMFormulaByFeaturesAndSwSystemconsts.feature ⌈For each FMFormulaByFeaturesAndSwSystemconsts the reference in the role feature shall exist.⌋*()*

The class FMFormulaByFeaturesAndSwSystemconsts uses the standard AUTOSAR formula language but extends it with features. That is, unlike SwSystemconstDependentFormula, which only allows references to SwSystemconsts, FMFormulaByFeaturesAndSwSystemconsts allows references to SwSystemconsts *and* FMFeatures.

[TPS_FMDT_00048] FMFormulaByFeaturesAndSwSystemconsts can refer to features and system constants ⌈A formula of class FMFormulaByFeaturesAndSwSystemconsts is an expression that can use both FMFeatures and SwSystemconsts.⌋(*RS_FMDT_00008, RS_FMDT_00010*)

7.2.4 FMConditionByFeaturesAndSwSystemconsts

Class	<<atpMixedString>> **FMConditionByFeaturesAndSwSystemconsts**
Package	M2::AUTOSARTemplates::FeatureModelTemplate
Note	A boolean expression that has the syntax of the AUTOSAR formula language and may use references to features or system constants as operands.

▽

△

Class	<<atpMixedString>> **FMConditionByFeaturesAndSwSystemconsts**			
Base	*ARObject*, *FMFormulaByFeaturesAndSwSystemconsts*, *FormulaExpression*, *SwSystemconstDependent Formula*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 7.4: FMConditionByFeaturesAndSwSystemconsts

[TPS_FMDT_00050] The result of `FMConditionByFeaturesAndSwSystemconsts` is interpreted as a boolean value. ⌈The result of a formula of class `FMConditionByFeaturesAndSwSystemconsts` shall be interpreted as a boolean value where 0 shall be interpreted as *false* and any value different from 0 shall be interpreted as *true*. This is the same approach as used by `ConditionByFormula`.⌋*(RS_FMDT_00008, RS_FMDT_00010)*

An element of class `FMConditionByFeaturesAndSwSystemconsts` is aggregated by `FMFeatureMapAssertion` in the role `fmSyscond` to define assertions for feature mappings.

7.2.5 Evaluating Expressions that use Features and Attributes

An expression that uses features or feature attributes can only be evaluated in the context of a `FMFeatureSelectionSet`. The reason is that in order to evaluate the expression, we need to substitute values for the references to features (1 if selected and 0 otherwise) and for the references to feature attributes (the default value or the one defined in the `FMFeatureSelection`). This information is only available as part of a `FMFeatureSelectionSet`.

First, we need to extend the definition of the *feature set* of a `FMFeatureSelectionSet` to add all features in the `include` and `FMFeatureSelectionSet`s:

[TPS_FMDT_00059] Definition of *recursive feature set* of a `FMFeatureSelectionSet` ⌈Let S be a `FMFeatureSelectionSet` that refers to `FMFeatureSelectionSet`s $\{S_1, S_2, \ldots, S_n\}$ in the role `include`.

Then the *recursive feature set* of S is defined as

$$\text{recursive feature set}(S) = \text{feature set}(S) \cup \bigcup_{S_i} \text{recursive feature set}(S_i)$$

⌋*(RS_FMDT_00003)*

Next, we define the `state` of a `FMFeature` in a `FMFeatureSelectionSet`. Again, this definition includes all `include` and `FMFeatureSelectionSet`s:

[TPS_FMDT_00058] Definition of *state* of a `FMFeature` in a `FMFeatureSelectionSet` ⌈Let f be a `FMFeature` and S be a `FMFeatureSelection` where f is in the *recursive feature set* of S. Then the *state* of f in S is defined as follows:

1. If f is in the *feature set* of S, let s be the `FMFeatureSelection` that refers to f in the role `feature`.

   Then the *state* of f in S is the value of the attribute `state` of s.

2. If f is not in the *feature set* of S, let $\{S_1, S_2, \ldots, S_n\}$ be the `FMFeatureSelectionSet`s that S refers to in the role `include`. Because f is in the *recursive feature set* of S, there shall be at least one S_i that defines the *state* of f.

   Then the *state* of f in S is the state of f in S_i.

⌋*(RS_FMDT_00003)*

Note that the second step in [TPS_FMDT_00058] retrieves a consistent result (i.e., no conflicting *states* such as `selected` and `deselected`) because of constraint [constr_5003] and constraint [constr_5025].

[TPS_FMDT_00057] Evaluating an Expression that uses Features and Attributes ⌈Let S be a `FMFeatureSelectionSet`. To evaluate an expression that uses attributes, the following steps shall be performed.

1. Replace all reference to `SwSystemconst`s by their values.
2. For each reference to a `FMFeature` f, we distinguish between two cases.
   (a) f is in the *recursive feature set* of S.
      i. If the *state* of f in S is `selected`, then the reference to f is replaced by the value 1.
      ii. If the *state* of f in S is `deselected`, then the reference to f is replaced by the value 0.
      iii. If the *state* of f in S is `undecided`, then this is considered an *error*.
   (b) f is *not* in the *recursive feature set* of S. This is considered an *error*.
3. For each reference to a `FMAttributeDef`,
   (a) If S aggregates a `FMFeatureSelection` s in the role `selection` that aggregates an `FMAttributeValue` v which refers to a in the role `definition`, then the reference is replaced by the contents of the attribute `value` of v.
   (b) Otherwise, let $\{S_1, S_2, \ldots, S_n\}$ be the `FMFeatureSelectionSet`s that S refers to in the role `include`. Repeat the previous step recursively for all S_i elements.
   (c) Otherwise, if a has a `defaultValue`, then the reference is replaced by the contents of the attribute `defaultValue` of a.
   (d) It is considered an error if none of the above steps can find a value.

⌋*(RS_FMDT_00008, RS_FMDT_00010)*

Note that when we look for the value of an attribute in [TPS_FMDT_00057], then do *not* look at the `state` of a `FMFeatureSelection`. That is, `FMAttributeValue` could also be taken from a `FMFeatureSelection` whose `state` is `deselected`. It is up to the party that creates the feature model, feature selection and feature mapping to decide if this is appropriate and eventually adapt the condition appropriately.

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 ([5]).

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`.

B Mentioned Class Tables

Class	***ARElement*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	An element that can be defined stand-alone, i.e. without being part of another element (except for packages of course).			
Base	*ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	AclObjectSet, AclOperation, AclPermission, AclRole, AliasNameSet, *AutosarDataType*, *BaseType*, BlueprintMappingSet, BuildActionManifest, CalibrationParameterValueSet, ClientIdDefinitionSet, Collection, CompuMethod, ConsistencyNeedsBlueprintSet, ConstantSpecification, ConstantSpecification MappingSet, CryptoServiceKey, CryptoServicePrimitive, CryptoServiceQueue, DataConstr, Data ExchangePoint, DataTransformationSet, DataTypeMappingSet, *DiagnosticCommonElement*, Diagnostic Connection, DiagnosticContributionSet, DltContext, DltEcu, Documentation, E2EProfileCompatibility Props, EndToEndProtectionSet, EthIpProps, EthTcpIpIcmpProps, EthTcpIpProps, EvaluatedVariantSet, FMFeature, FMFeatureMap, FMFeatureModel, FMFeatureSelectionSet, FunctionGroupSet, General PurposeConnection, HwCategory, HwElement, HwType, IPSecConfigProps, *IdsCommonElement*, Ids Design, InterpolationRoutineMappingSet, KeywordSet, LifeCycleInfoSet, LifeCycleStateDefinitionGroup, LogAndTraceMessageCollectionSet, McFunction, McGroup, ModeDeclarationGroup, ModeDeclaration MappingSet, PhysicalDimension, PhysicalDimensionMappingSet, *PlatformModuleEndpointConfiguration*, *PortInterface*, PortInterfaceMappingSet, PortPrototypeBlueprint, PostBuildVariantCriterion, PostBuild VariantCriterionValueSet, PredefinedVariant, RapidPrototypingScenario, SdgDef, SignalService TranslationPropsSet, SoftwareCluster, SomeipSdClientEventGroupTimingConfig, SomeipSdClient ServiceInstanceConfig, SomeipSdServerEventGroupTimingConfig, SomeipSdServerServiceInstance Config, SwAddrMethod, SwAxisType, *SwComponentType*, SwRecordLayout, SwSystemconst, Sw SystemconstantValueSet, System, SystemSignal, SystemSignalGroup, *TimingExtension*, TlvDataId DefinitionSet, TransformationPropsSet, Unit, UnitGroup, ViewMapSet			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table B.1: ARElement

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	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**AUTOSAR**			
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:** atp.Splitkey=arPackage.shortName, arPackage.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
fileInfoComment	FileInfoComment	0..1	aggr	This represents a possibility to provide a structured comment in an AUTOSAR file. **Stereotypes:** atpStructuredComment **Tags:** xml.roleElement=true xml.sequenceOffset=-10 xml.typeElement=false
introduction	DocumentationBlock	0..1	aggr	This represents an introduction on the Autosar file. It is intended for example to represent disclaimers and legal notes. **Tags:**xml.sequenceOffset=20

Table B.2: AUTOSAR

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

▽

△

Class	**AdminData**			
language	LEnum	0..1	attr	This attribute specifies the master language of the document or the document fragment. The master language is the one in which the document is maintained and from which the other languages are derived from. In particular in case of inconsistencies, the information in the master language is priority. **Tags:**xml.sequenceOffset=20
sdg	Sdg	*	aggr	This property allows to keep special data which is not represented by the standard model. It can be utilized to keep e.g. tool specific data. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=sdg, sdg.variationPoint.shortLabel xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=60 xml.typeElement=false xml.typeWrapperElement=false
usedLanguages	MultiLanguagePlainText	0..1	aggr	This property specifies the languages which are provided in the document. Therefore it should only be specified in the top level admin data. For each language provided in the document there is one entry in MultilanguagePlain Text. The content of each entry can be used for illustration of the language. The used language itself depends on the language attribute in the entry. **Tags:**xml.sequenceOffset=30

Table B.3: AdminData

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

Table B.4: ConditionByFormula

Class	<<atpMixedString>> ***FormulaExpression*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::FormulaLanguage
Note	This class represents the syntax of the formula language. The class is modeled as an abstract class in order to be specialized into particular use cases. For each use case the referable objects might be specified in the specialization.

▽

△

Class	<<atpMixedString>> ***FormulaExpression*** (abstract)			
Base	*ARObject*			
Subclasses	CompuGenericMath, *FMFormulaByFeaturesAndAttributes*, *SwSystemconstDependentFormula*, TDEvent OccurrenceExpressionFormula, TimingConditionFormula			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
atpReference	Referrable	*	ref	The referable object shall yield a numerical / boolean value. **Stereotypes:** atpAbstract
atpString Reference	Referrable	*	ref	The referable object shall yield a string value. **Stereotypes:** atpAbstract

Table B.5: FormulaExpression

Class	***Identifiable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.			
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstanceFilter*, *AbstractServiceInstance*, *AdaptiveModuleInstantiation*, ApplicationEndpoint, ApplicationError, ArtifactChecksum, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, BlockState, *BuildActionEntity*, BuildActionEnvironment, Chapter, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, *CouplingPortStructuralElement*, CryptoKeySlot, *CryptoServiceMapping*, DataPrototypeGroup, DataTransformation, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, DiagnosticFunctionInhibitSource, *DiagnosticRoutineSubfunction*, DltApplication, DltArgument, DltMessage, DoIpInterface, DoIpLogicAddress, DoIpRoutingActivation, EndToEndProtection, EthernetWakeupSleepOnDatalineConfig, EventHandler, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FlexrayArTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, InternalTriggeringPoint, Keyword, LifeCycleState, Linker, MacMulticastGroup, McDataInstance, MemorySection, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, *NmNode*, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, ResourceConsumption, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecureCommunicationAuthenticationProps, SecureCommunicationFreshnessProps, SecurityEventContextProps, *ServiceNeeds*, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SomeipTpChannel, *SpecElementReference*, *StackUsage*, StaticSocketConnection, StructuredReq, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SystemMapping, *TimeBaseResource*, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, *TransformationProps*, TransformationTechnology, Trigger, VariableAccess, VariationPointProxy, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***Identifiable*** (abstract)			
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:**xml.sequenceOffset=-25
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:**xml.sequenceOffset=-50
desc	MultiLanguageOverview Paragraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:**xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:**xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:**xml.attribute=true

Table B.6: Identifiable

Class	***MultilanguageReferrable*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable

▽

△

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

Table B.7: MultilanguageReferrable

Class	***PortPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for the ports of an AUTOSAR software component. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports.			
Base	*ARObject*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*AbstractProvidedPortPrototype*, *AbstractRequiredPortPrototype*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
logAndTraceMessageCollectionSet	LogAndTraceMessageCollectionSet	0..1	ref	Reference to a collection of Log or Trace messages that will be used by the application. **Tags:**atp.Status=draft

Table B.8: PortPrototype

Primitive	**PositiveInteger**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is a positive integer which can be denoted in decimal, binary, octal and hexadecimal. The value is between 0 and 4294967295. **Tags:** xml.xsd.customType=POSITIVE-INTEGER xml.xsd.pattern=0|[\+]?[1-9][0-9]*|0[xX][0-9a-fA-F]+|0[bB][0-1]+|0[0-7]+ xml.xsd.type=string

Table B.9: PositiveInteger

Class	**PostBuildVariantCondition**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies the value which shall be assigned to a particular variant criterion in order to bind the variation point. If multiple criterion/value pairs are specified, they shall all match to bind the variation point. In other words binding can be represented by (criterion1 == value1) && (condition2 == value2) ...			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**PostBuildVariantCondition**			
matching Criterion	PostBuildVariant Criterion	1	ref	This is the criterion which needs to match the value in order to make the PostbuildVariantCondition to be true.
value	Integer	1	attr	This is the particular value of the post-build variant criterion. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table B.10: PostBuildVariantCondition

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

Table B.11: PostBuildVariantCriterion

Class	**PostBuildVariantCriterionValue**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies the value which shall be assigned to a particular variant criterion in order to bind the variation point. If multiple criterion/value pairs are specified, they all shall match to bind the variation point.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
annotation	Annotation	*	aggr	This provides the ability to add information why the value is set like it is. **Tags:**xml.sequenceOffset=30
value	Integer	1	attr	This is the particular value of the post-build variant criterion. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
variantCriterion	PostBuildVariant Criterion	1	ref	This association selects the variant criterion whose value is specified. **Tags:**xml.sequenceOffset=10

Table B.12: PostBuildVariantCriterionValue

Class	**PredefinedVariant**
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling

▽

△

Class	**PredefinedVariant**			
Note	This specifies one predefined variant. It is characterized by the union of all system constant values and post-build variant criterion values aggregated within all referenced system constant value sets and post build variant criterion value sets plus the value sets of the included variants. **Tags:**atp.recommendedPackage=PredefinedVariants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
includedVariant	PredefinedVariant	*	ref	The associated variants are considered part of this PredefinedVariant. This means the settings of the included variants are included in the settings of the referencing PredefinedVariant. Nevertheless the included variants might be included in several predefined variants.
postBuildVariant CriterionValue Set	PostBuildVariant CriterionValueSet	*	ref	This is the postBuildVariantCriterionValueSet contributing to the predefinded variant.
sw Systemconstant ValueSet	SwSystemconstant ValueSet	*	ref	This ist the set of Systemconstant Values contributing to the predefined variant.

Table B.13: PredefinedVariant

Class	***Referrable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, Bsw VariableAccess, CouplingPortTrafficClassAssignment, *DiagnosticEnvModeElement*, EthernetPriority Regeneration, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, ModeTransition, *MultilanguageReferrable*, PncMappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, Socket ConnectionBundle, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:**xml.sequenceOffset=-90

Table B.14: Referrable

Class	**Sdg**
Package	M2::MSR::AsamHdo::SpecialData

▽

△

Class	Sdg			
Note	Sdg (SpecialDataGroup) is a generic model which can be used to keep arbitrary information which is not explicitly modeled in the meta-model. Sdg can have various contents as defined by sdgContentsType. Special Data should only be used moderately since all elements should be defined in the meta-model. Thereby SDG should be considered as a temporary solution when no explicit model is available. If an sdg Caption is available, it is possible to establish a reference to the sdg structure.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
gid	NameToken	1	attr	This attributes specifies an identifier. Gid comes from the SGML/XML-Term "Generic Identifier" which is the element name in XML. The role of this attribute is the same as the name of an XML - element. **Tags:**xml.attribute=true
sdgCaption	SdgCaption	0..1	aggr	This aggregation allows to assign the properties of Identifiable to the sdg. By this, a shortName etc. can be assigned to the Sdg. **Tags:**xml.sequenceOffset=20
sdgContents Type	SdgContents	0..1	aggr	This is the content of the Sdg. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table B.15: Sdg

Class	SwComponentPrototype			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	Role of a software component within a composition.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
type	SwComponentType	0..1	tref	Type of the instance. **Stereotypes:** isOfType

Table B.16: SwComponentPrototype

Class	SwSystemconst
Package	M2::MSR::DataDictionary::SystemConstant
Note	This element defines a system constant which serves an input to select a particular variation point. In particular a system constant serves as an operand of the binding function (swSyscond) in a Variation point. Note that the binding process can only happen if a value was assigned to to the referenced system constants. **Tags:**atp.recommendedPackage=SwSystemconsts
Base	*ARElement*, *ARObject*, *AtpDefinition*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*

▽

△

Class	**SwSystemconst**			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
swDataDef Props	SwDataDefProps	0..1	aggr	This denotes the data definition properties of the system constant. This supports to express the limits and optionally a conversion within the internal to physical values by a compu method. **Tags:**xml.sequenceOffset=40

Table B.17: SwSystemconst

Class	<<atpMixedString>> ***SwSystemconstDependentFormula*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents an expression depending on system constants.			
Base	*ARObject*, *FormulaExpression*			
Subclasses	*AttributeValueVariationPoint*, BlueprintFormula, ConditionByFormula, *FMFormulaByFeaturesAndSw Systemconsts*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
sysc	SwSystemconst	0..1	ref	This refers to a system constant. The internal (coded) value of the system constant shall be used. **Tags:**xml.sequenceOffset=50
syscString	SwSystemconst	0..1	ref	syscString indicates that the referenced system constant shall be evaluated as a string according to [TPS_SWCT_01431].

Table B.18: SwSystemconstDependentFormula

Class	**SwSystemconstValue**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class assigns a particular value to a system constant.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
annotation	Annotation	*	aggr	This provides the ability to add information why the value is set like it is. **Tags:**xml.sequenceOffset=30
swSystemconst	SwSystemconst	1	ref	This is the system constant to which the value applies. **Tags:**xml.sequenceOffset=10
value	Numerical	1	attr	This is the particular value of a system constant. It is specified as Numerical. Further restrictions may apply by the definition of the system constant. The value attribute defines the internal value of the Sw Systemconst as it is processed in the Formula Language. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20

Table B.19: SwSystemconstValue

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 postBuildVariant Criterion is fulfilled.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
blueprint Condition	DocumentationBlock	0..1	aggr	This represents a description that documents how the variation point shall be resolved when deriving objects from the blueprint. Note that variationPoints are not allowed within a blueprintCondition. **Tags:**xml.sequenceOffset=28
desc	MultiLanguageOverview Paragraph	0..1	aggr	This allows to describe shortly the purpose of the variation point. **Tags:**xml.sequenceOffset=20
formalBlueprint Generator	BlueprintGenerator	0..1	aggr	This represents a description that documents how the variation point shall be resolved when deriving objects from the blueprint by using ARMQL. Note that variationPoints are not allowed within a formal BlueprintGenerator. **Tags:** atp.Status=draft xml.sequenceOffset=30
postBuildVariant Condition	PostBuildVariant Condition	*	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	attr	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. **Stereotypes:** atpIdentityContributor **Tags:**xml.sequenceOffset=10
swSyscond	ConditionByFormula	0..1	aggr	This condition acts as Binding Function for the Variation Point. Note that the multiplicity is 0..1 in order to support pure postBuild variants. **Tags:**xml.sequenceOffset=30

Table B.20: VariationPoint

C Constraint History

C.1 Change History for AUTOSAR R4.1.1

C.1.1 Added Constraints R4.1.1

Id	Heading
[constr_5001]	FMFeatureRelation shall not establish self-references
[constr_5002]	FMFeatureSelectionSet shall not have cycles in the include relation
[constr_5003]	FMFeatureSelectionSet shall not overwrite the state of included features
[constr_5005]	FMFeature shall not be referenced from more than one FMFeatureDecomposition
[constr_5007]	FMFeature shall only be referenced from one FMFeatureModel in the role feature
[constr_5008]	If present, the root feature shall be part of the feature model
[constr_5009]	Root feature shall be present if and only if the feature model is not empty
[constr_5010]	FMFeatureDecomposition may refer to a root feature of another feature model, but only once.
[constr_5011]	FMFormulaByFeaturesAndAttributes can refer to FMFeatures and FMAttributeDefs, but not to system constants
[constr_5013]	Attributes min and max of FMFeatureDecomposition reserved for category MULTIPLEFEATURE
[constr_5018]	FMFeatureSelectionSet shall not include the same feature twice
[constr_5019]	FMFeatureModel shall not contain the same FMFeature twice
[constr_5020]	Every FMFeature shall be contained in a FMFeatureModel
[constr_5021]	The underlying graph of a feature model shall be a tree.
[constr_5022]	The root feature of a FMFeatureModel refers to the root of the underlying tree.
[constr_5023]	FMFeatureSelectionSet may only refer to FMFeatures from the associated FMFeatureModel
[constr_5024]	FMFeatureSelectionSet shall not include itself
[constr_5025]	Multiple include in FMFeatureSelectionSet shall be consistent
[constr_5026]	Semantics of attributes max and min in class FMAttributeDef
[constr_5027]	Semantics of attributes max and min of FMAttributeDef in class FMAttributeValue
[constr_5028]	Only one FMAttributeValue per FMAttributeDef

Table C.1: changed Constraints in 4.1.1

C.1.2 Changed Constraints R4.1.1

none

C.1.3 Deleted Constraints R4.1.1

none

C.1.4 Added Traceables R4.1.1

Id	Heading
[TPS_FMDT_00001]	Feature Models may be empty
[TPS_FMDT_00002]	Definition of *Feature*
[TPS_FMDT_00003]	Definition of *Feature Selection*
[TPS_FMDT_00004]	Definition of *Feature Model*
[TPS_FMDT_00005]	Definition of *Product Model*
[TPS_FMDT_00006]	Definition of *Product Line Model*
[TPS_FMDT_00007]	Definition of *Product*
[TPS_FMDT_00008]	Definition of *Product Line*
[TPS_FMDT_00009]	Definition of *Feature Set* of a FMFeatureSelectionSet
[TPS_FMDT_00012]	Default values for attributes min and max
[TPS_FMDT_00013]	Feature Models are optional
[TPS_FMDT_00014]	Definition of *Parent Feature*, *Child Feature*
[TPS_FMDT_00015]	MANDATORYFEATURE
[TPS_FMDT_00016]	OPTIONALFEATURE
[TPS_FMDT_00017]	ALTERNATIVEFEATURE
[TPS_FMDT_00018]	MULTIPLEFEATURE
[TPS_FMDT_00019]	Predefined values for the category of FMFeatureRelation
[TPS_FMDT_00020]	Structure of FMFeatureRelation
[TPS_FMDT_00021]	category attribute of FMFeatureRelation
[TPS_FMDT_00023]	Extensibility of category attribute of FMFeatureRelation
[TPS_FMDT_00024]	Attributes maximumIntendedBindingTime and minimumIntended-BindingTime are only a hint
[TPS_FMDT_00025]	Set of *affected variation points* for a FMFeatureMapElement
[TPS_FMDT_00030]	Definition of *Valid Feature Selection*
[TPS_FMDT_00032]	*Inclusion graph* for FMFeatureSelectionSets
[TPS_FMDT_00033]	Special data for feature models
[TPS_FMDT_00034]	Definition of *Underlying Graph* of a FMFeatureModel
[TPS_FMDT_00035]	Definition of *Features* of a FMFeatureModel
[TPS_FMDT_00036]	Definition of *Root Feature* of a FMFeatureModel
[TPS_FMDT_00037]	Semantics of FMFeatureMapElement
[TPS_FMDT_00038]	Definition of *Affected Variation Points* for a FMFeature
[TPS_FMDT_00039]	Name of a FMFeature
[TPS_FMDT_00040]	Description for a FMFeature
[TPS_FMDT_00041]	Purpose of FMFeatureDecomposition
[TPS_FMDT_00042]	Purpose of FMFeature
[TPS_FMDT_00043]	Purpose of FMFeatureModel
[TPS_FMDT_00044]	Semantics of FMFeatureRelation
[TPS_FMDT_00045]	Semantics of FMFeatureRestriction
[TPS_FMDT_00046]	Semantics of FMFeatureDecomposition
[TPS_FMDT_00047]	Feature models are splitable
[TPS_FMDT_00048]	FMFormulaByFeaturesAndSwSystemconsts can refer to features and system constants
[TPS_FMDT_00049]	The result of FMConditionByFeaturesAndAttributes is interpreted as a boolean value.
[TPS_FMDT_00050]	The result of FMConditionByFeaturesAndSwSystemconsts is interpreted as a boolean value.
[TPS_FMDT_00051]	Purpose of FMAttributeDef
[TPS_FMDT_00052]	Identifying FMFeatureRelations
[TPS_FMDT_00053]	Semantics of FMAttributeValue
[TPS_FMDT_00054]	Semantics of attributes minimumIntendedBindingTime and maximu-mIntendedBindingTime

[TPS_FMDT_00055]	Semantics of `minimumSelectedBindingTime` and `maximumSelected-BindingTime`
[TPS_FMDT_00056]	`minimumSelectedBindingTime` and `maximumSelectedBindingTime` are only hints
[TPS_FMDT_00057]	Evaluating an Expression that uses Features and Attributes
[TPS_FMDT_00058]	Definition of *state* of a `FMFeature` in a `FMFeatureSelectionSet`
[TPS_FMDT_00059]	Definition of *recursive feature set* of a `FMFeatureSelectionSet`
[TPS_FMDT_00060]	Purpose of `FMFeatureSelectionSet`
[TPS_FMDT_00061]	Documenting `FMFeatureRelation`s
[TPS_FMDT_00062]	Identifying `FMFeatureRestriction`s
[TPS_FMDT_00063]	Documenting `FMFeatureRestriction`s

Table C.2: changed Constraints in 4.1.1

C.1.5 Changed Traceables R4.1.1

none

C.1.6 Deleted Traceables R4.1.1

none

C.2 Change History for AUTOSAR R4.2.1 against R4.1.3

C.2.1 Added Constraints in 4.2.1

none

C.2.2 Changed Constraints in 4.2.1

none

C.2.3 Deleted Constraints in 4.2.1

none

C.2.4 Added Traceables in 4.2.1

Id	Heading
[TPS_FMDT_00064]	Usage of *Life Cycle*

Table C.3: Added Traceables in 4.2.1

C.2.5 Changed Traceables in 4.2.1

none

C.2.6 Deleted Traceables in 4.2.1

none

C.3 Change History for AUTOSAR R4.2.2 against R4.2.1

C.3.1 Added Constraints in 4.2.2

none

C.3.2 Changed Constraints in 4.2.2

none

C.3.3 Deleted Constraints in 4.2.2

none

C.3.4 Added Traceables in 4.2.2

none

C.3.5 Changed Traceables in 4.2.2

none

C.3.6 Deleted Traceables in 4.2.2

none

C.4 Change History for AUTOSAR R4.3.0 against R4.2.2

C.4.1 Added Constraints in 4.3.0

none

C.4.2 Changed Constraints in 4.3.0

none

C.4.3 Deleted Constraints in 4.3.0

none

C.4.4 Added Traceables in 4.3.0

none

C.4.5 Changed Traceables in 4.3.0

none

C.4.6 Deleted Traceables in 4.3.0

none

C.5 Change History for AUTOSAR R4.3.1 against R4.3.0

C.5.1 Added Constraints in 4.3.1

none

C.5.2 Changed Constraints in 4.3.1

none

C.5.3 Deleted Constraints in 4.3.1

none

C.5.4 Added Traceables in 4.3.1

none

C.5.5 Changed Traceables in 4.3.1

none

C.5.6 Deleted Traceables in 4.3.1

none

C.6 Change History for AUTOSAR R4.4.0 against R4.3.1

C.6.1 Added Constraints in 4.4.0

none

C.6.2 Changed Constraints in 4.4.0

none

C.6.3 Deleted Constraints in 4.4.0

none

C.6.4 Added Traceables in 4.4.0

none

C.6.5 Changed Traceables in 4.4.0

none

C.6.6 Deleted Traceables in 4.4.0

none

C.7 Change History for AUTOSAR R19-11 against R4.4.0

C.7.1 Added Constraints in 19-11

none

C.7.2 Changed Constraints in 19-11

none

C.7.3 Deleted Constraints in 19-11

none

C.7.4 Added Traceables in 19-11

none

C.7.5 Changed Traceables in 19-11

none

C.7.6 Deleted Traceables in 19-11

none

C.8 Change History for AUTOSAR R20-11 against R19-11

C.8.1 Added Constraints in R20-11

none

C.8.2 Changed Constraints in R20-11

none

C.8.3 Deleted Constraints in R20-11

none

C.8.4 Added Traceables in R20-11

none

C.8.5 Changed Traceables in R20-11

none

C.8.6 Deleted Traceables in R20-11

none

C.9 Change History for AUTOSAR R21-11 against R20-11

C.9.1 Added Constraints in R21-11

Number	Heading
[constr_3657]	Multiplicity of `FMAttributeDef.max` and `FMAttributeDef.min`
[constr_3658]	Multiplicity of `FMFeatureDecomposition.category`
[constr_3659]	Multiplicity of `FMFeatureDecomposition.feature`
[constr_3660]	Multiplicity of `FMFeatureRelation.feature`

▽

△

Number	Heading
[constr_3661]	Multiplicity of FMFeatureSelection.feature
[constr_3662]	Multiplicity of FMFeatureSelection.state
[constr_3663]	Multiplicity of FMAttributeValue.definition
[constr_3664]	Multiplicity of FMAttributeValue.value
[constr_3665]	Multiplicity of FMFormulaByFeaturesAndAttributes.attribute
[constr_3666]	Multiplicity of FMFormulaByFeaturesAndAttributes.feature
[constr_3667]	Multiplicity of FMFormulaByFeaturesAndSwSystemconsts.feature

Table C.4: Added Constraints in R21-11

C.9.2 Changed Constraints in R21-11

none

C.9.3 Deleted Constraints in R21-11

none

C.9.4 Added Traceables in R21-11

none

C.9.5 Changed Traceables in R21-11

none

C.9.6 Deleted Traceables in R21-11

none