VEC Tutorial

Examples for the proper use of the VEC Model

Version 1.4, 12.05.2016

Status: Released by Project Group “Car Electric”

Notice: This is a printable Version of the Online Tutorials published at: http://ecad-wiki.prostep.org. The newest version can be found there.

Abstract

The VDA recommendation 4968 “Vehicle Electric Container (VEC)” defines an information model, a data dictionary, and an XML schema derived from and compliant to the model.

The intention of the model was to cover a wide range of use cases and application scenarios. For this reason the specification have to be kept generic in some degree.

To avoid dialects in VEC implementations further guidelines or recommendations are necessary. This document contributes to the unambiguous interpretation of the VEC standard. For various wiring harness definition or electrical system aspects the correct instantion is shown.

Document History

Version	Date	Author	Description
1.0	05.04.2013	Johannes Becker	First release; Compliant to VEC 1.0.5
1.1	29.07.2014	Johannes Becker	Adapted Tutorials to VEC Version V1.1.0
1.2	05.08.2014	Johannes Becker	Added the Tutorials 3.5 Connectors and 3.7 Accessories
1.3	29.10.2014	Johannes Becker	Adapted the Tutorials for the new placement definition in VEC 1.1. Added Tutorials 3.8 Grommets with Placement and 3.9 Channels and extended the example 3.5 Connectors for connectors with multiple segment connection points. Added Tutorial 1.3 Physical Properties
1.4	12.05.2016	Johannes Becker	Added or extended multiple requested tutorials: 3.5 Connectors 3.6 ECUs and EE Components 3.7 Pinning 3.11 Fixings 4.1 Placements For detailed release notes, please see ProSTEP JIRA

Table of content

1 Key Concepts

1.1 Parts and Documents

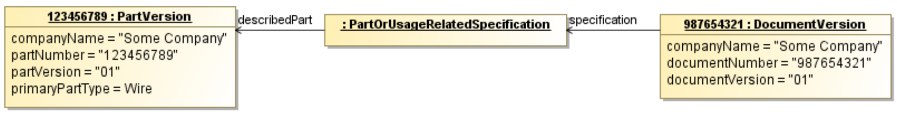

123456789 : PartVersion
companyName = "Some Company"
partNumber = "123456789"
partVersion = "01"
primaryPartType = Wire
describedPart
: PartOrUsageRelatedSpecification
specification
987654321 : DocumentVersion
companyName = "Some Company"
documentNumber = "987654321"
documentVersion = "01"

Figure 1: Parts and Documents

The figure above illustrates one of the most fundamental concepts of the VEC – the separation of a part and its description. In the VEC a part (PartVersion) does not contain any information about the part, except its PDM Information (PartNumber, PartVersion…). All the information about the technical properties of a part is expressed by a subclass of PartOrUsageRelatedSpecifications (e.g. a WireSpecification). The specifications itsselfes are contained in a DocumentVersion.

This approach enables the VEC to address two scenarios properly:

- The description of a part is changed, but the part itself is not changed (rereleased). This can happen for example if the actual technical properties of the part stay the same, but the description is extended or corrected.
- A document and the contained specifications are describing more than one part (e.g. a drawing for a certain class of terminals). In this case it can happen that the document and the specifications are changed, but not all of the described parts have to be changed (rereleased).

1.2 Usage Nodes

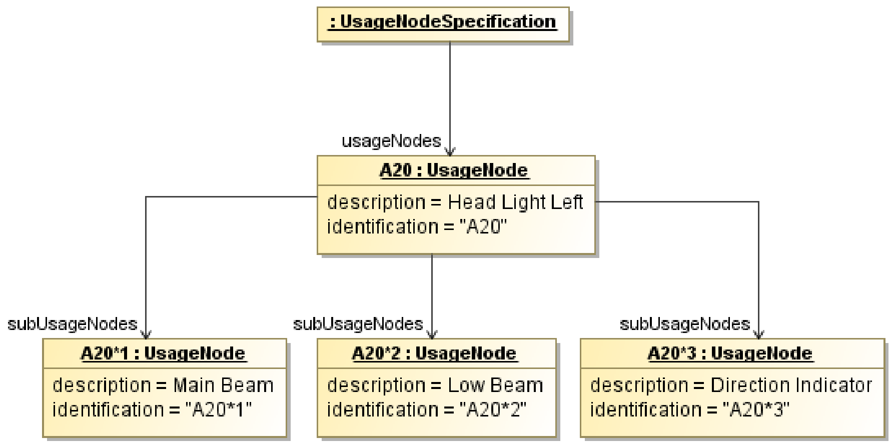


Figure 2: Usage Nodes

The example illustrates the use of *UsageNodes*. A *UsageNode* represents a position in an abstract vehicle. For example the Head Light Left. *UsageNodes* belong to the master data and they are defined on some company wide level. They can be used to enforce consistent naming over different projects and different development streams (e.g. between geometry and electrologic).

A *UsageNode* can be subdivided into more detailed *UsageNodes*. For example the Head Light can be split up into Main Beam, Low Beam and Direction Indicator.

1.3 *Physical Properties*

This section describes the different methods for specifying physical properties. The following contains the different methods / types of how values can be defined.

1.3.1 Numerical Values

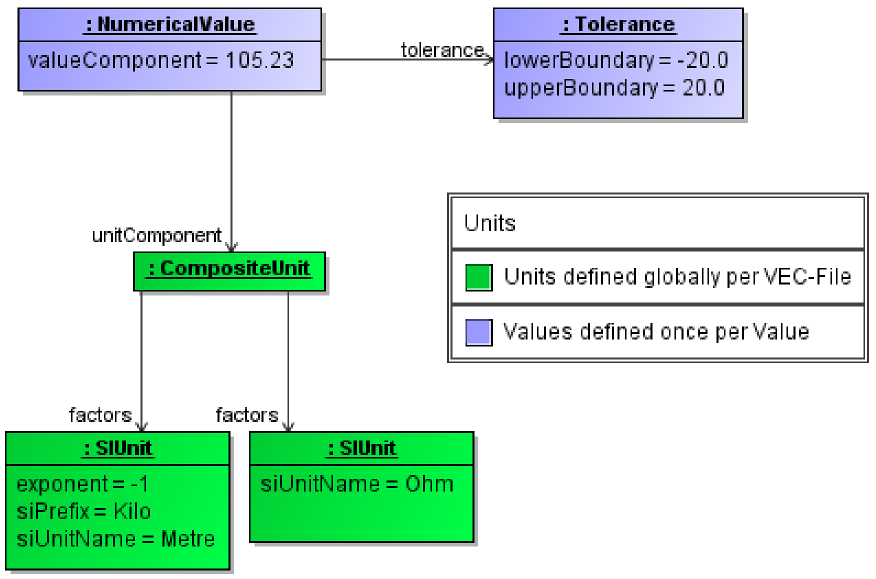


Figure 3: Numerical Values

Since most *NumericalValues* in the VEC are definitions for technical properties, each *NumericalValue* can reference a *Tolerance*. In the example, the specified value of '105.23' has a tolerance of +/- 20.0. Units for *NumericalValue*s are defined globally in the VEC and are reused for each *NumericalValue*. The example shows the definition of the *CompositeUnit* Ohm per Kilometre.

1.3.2 Reference Systems

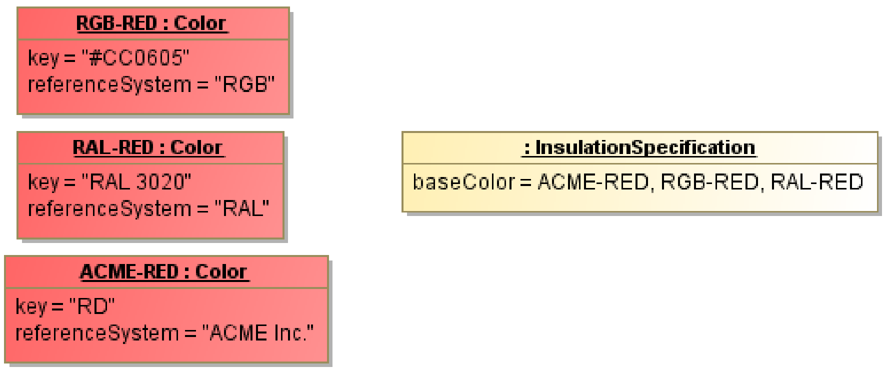


Figure 4: Reference Systems

This tutorial demonstrates how values with reference systems shall be used. In many cases (e.g. Colors) there is no single way to express a certain literal, but many different ways.

In order to correctly express such values the VEC gives the possibility to define not only the value, but the reference system in which the value is valid, too. This means if I have three valid ways to express the Color "Red", then I can define and differentiate them. If the value is defined in some standard reference system I will use this (e.g. RGB or RAL for colors). If the value is defined in some company specific reference system, this can be defined, too (see ACME Inc.). For attributes like the "baseColor" of a wire insulation it is possible to define the single value in different reference systems (in the example the color in RGB, RAL and company specific ACME Inc.). However all given values shall refer to the same "real" value.

2 PDM Information

2.1 Document Meta-Information

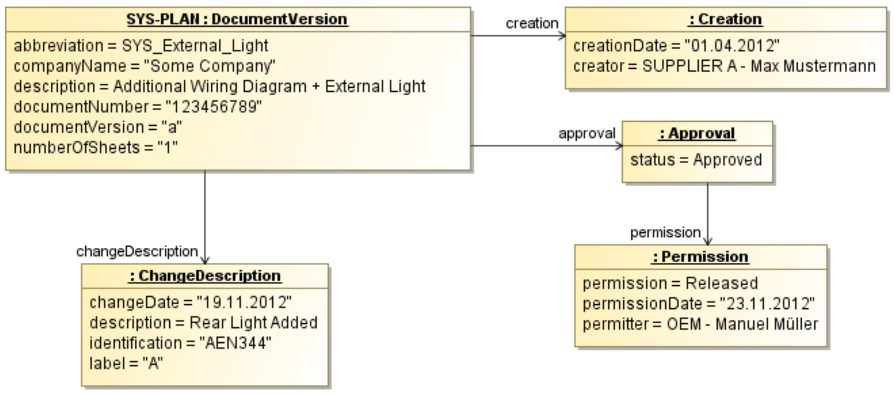

SYS-PLAN : DocumentVersion
abbreviation = SYS_External_Light
companyName = "Some Company"
description = Additional Wiring Diagram + External Light
documentNumber = "123456789"
documentVersion = "a"
numberOfSheets = "1"
creation
: Creation
creationDate = "01.04.2012"
creator = SUPPLIER A - Max Mustermann
approval
: Approval
status = Approved
permission
: Permission
permission = Released
permissionDate = "23.11.2012"
permitter = OEM - Manuel Müller
changeDescription
: ChangeDescription
changeDate = "19.11.2012"
description = Rear Light Added
identification = "AEN344"
label = "A"

Figure 5: Document Meta-Information

All information actually defined in a VEC file is contained in a DocumentVersion. Additionally this DocumentVersion carries all the meta-information about the underlying document (e.g. for a system schematic), This information is normally shown in drawings as a title block.

The DocumentVersion contains the information about the unique naming, a multilingual description, the DocumentNumber and so on. It has subelements to define the change history (ChangeDescription), the creation and the different approvals.

2.2 Item History

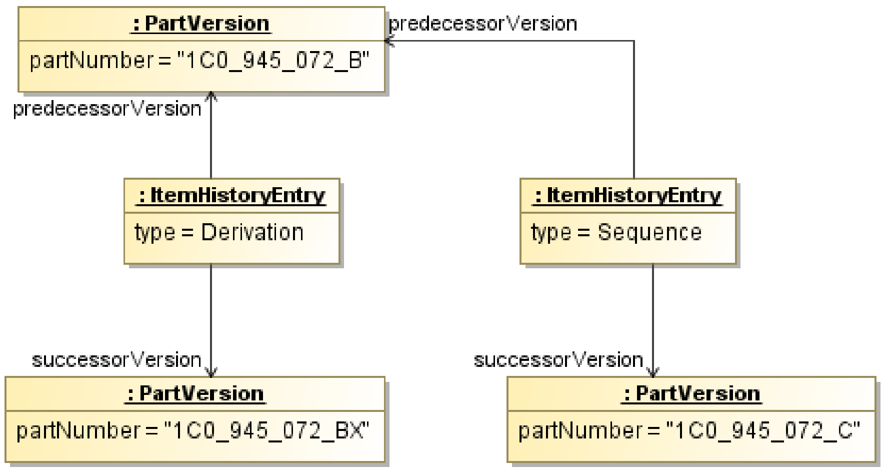


Figure 6: Item History

This example demonstrates how chronological relationships between PartVersion can be established. The VEC offers two types of relationships:

1. **Derivation:** Derivation means that the successor of the relationship is a newly developed part (variant) based on an existing part.
2. **Sequence:** Sequence means that the successor is a replacement for the predecessor.

3 Description & Instances of Components

In the next sections some examples for the specification of parts / components are presented. In order to keep the examples as simple as possible, the described PartVersions and the containing DocumentVersions are left out. Nevertheless they are needed for a real instantiation of the VEC.

3.1 Basic Component Description

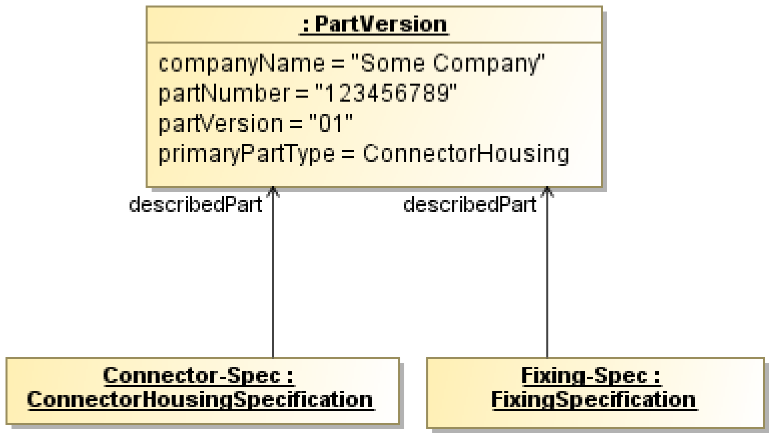

: PartVersion
companyName = "Some Company"
partNumber = "123456789"
partVersion = "01"
primaryPartType = ConnectorHousing
describedPart
describedPart
Connector-Spec : ConnectorHousingSpecification
Fixing-Spec : FixingSpecification

Figure 7: Basic Component Description

Many of the classes in the VEC are used for the defintion of the general properties of a component (Part Master Data). This is done with specifications (VEC Naming Convention “*XXXSpecification*”). A specification defines all properties of a component that are that are independant of a specific usage of the component (e.g. the number of cavities of connector or the cross section area of a wire). The type of the component, and thereby the available properties, is defined by the classification of the specification (e.g. *ConnectorHousingSpecification* for a connector housing). The PDM Information of a component (e.g. the part number) is defined by a *PartVersion*.

Since a component can have properties of different classifications (e.g. a connector can also have properties of a fixing) it is possible in the VEC to define multiple different specifications for a single part. The attribute "*primaryPartType*" defines which the primary character of the component is.

3.2 Instantiation of Parts

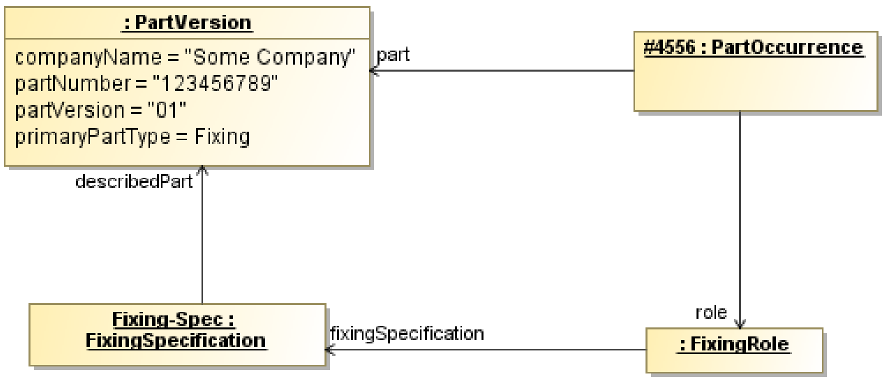


Figure 8: Instantiation of Parts

If a component should be used in the VEC it must be instantiated. This can be done in two ways. The first way is used if a concrete part number is known. The second way is used if a part number is unknown and only requirements for a component should be defined.

The first case is illustrated in this diagram and is quite similar to the way how components have been used in the KBL. The part is identified by a PartVersion with a PartNumber, Version and the issuing Company. The general properties of the component are defined by a specification (a FixingSpecification in the example). The use of the component is expressed by a "PartOccurrence". All usage specific properties of component (e.g. the length of a wire) are defined by a Role. The used role must correspond to a specification. The VEC Naming Convention is XXXRole corresponds to XXXSpecification.

In object oriented terms, one can say the PartVersion and Specification are defining the type (or classification) of a component, the PartOccurrence and the Role are defining the instance of a component.

3.3 Instantiation of Specifications without Part Number

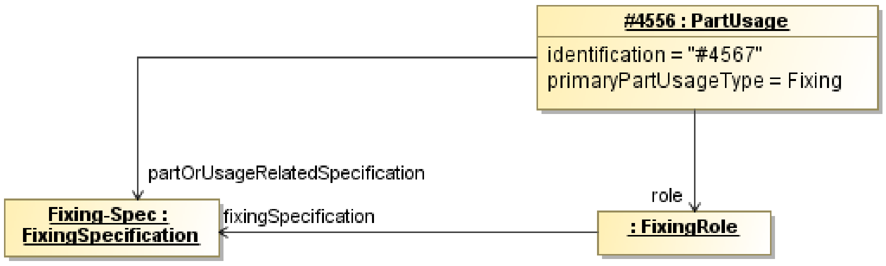

#4556 : PartUsage
identification = "#4567"
primaryPartUsageType = Fixing
partOrUsageRelatedSpecification
role
Fixing-Spec : FixingSpecification
fixingSpecification
: FixingRole

Figure 9: Instantiation of Specifications without Part Number

As mentioned in the example before, there are use cases where it is necessary to define properties of component (instance), without having a concrete part number. This is quite common for example in the electrological design, where for example the cross section area of a wire is specified, but the colouring of the wire is not yet known. In these cases the requirements for component have to be defined without having a concrete *PartVersion*.

For these cases, the VEC offers the concept of *PartUsages*. A *PartUsage* behaves like a *PartOccurrence* (it has instance specific attributes and can define *Roles*), but it is not necessary to reference a PartVersion. Instead it is possible to directly reference the needed specifications, which represent the requirements on the concrete component, selected later in the process.

3.4 Wires

3.4.1 Single Core Specification

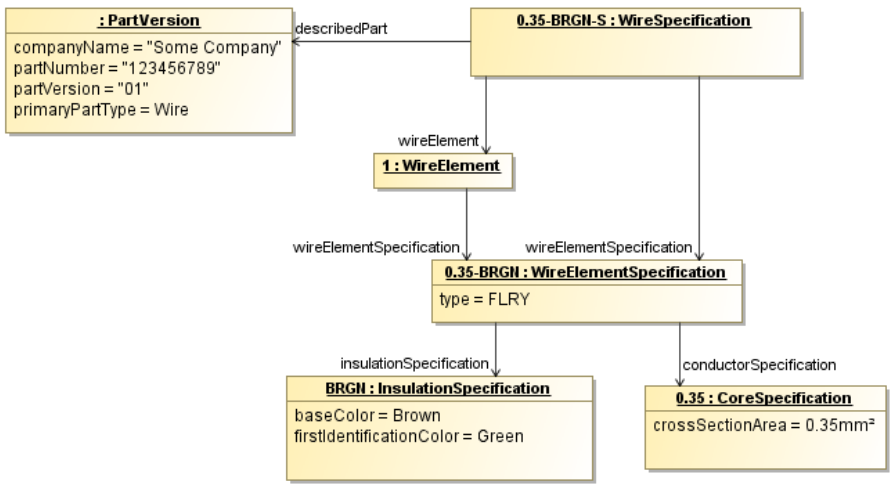


Figure 10: Single Core Specification

The figure above illustrates the specification of a single core wire. The WireSpecification is the PartOrUsageRelatedSpecification describing the PartVersion. In order to make it possible to reuse the specification of the different WireElements (see Multi Core), the actual definition of the structure of the wire is delegated to a WireElementSpecification. The WireElement defines the context specific identification of a WireElementSpecification in the context of a specific WireSpecification (mainly needed for multi cores, but due to a consistent modeling approach also necessary for single cores). The WireElement is used for reference when a WireSpecification is instantiated (e.g. by a PartOccurrence).

3.4.2 Multi Core Illustration

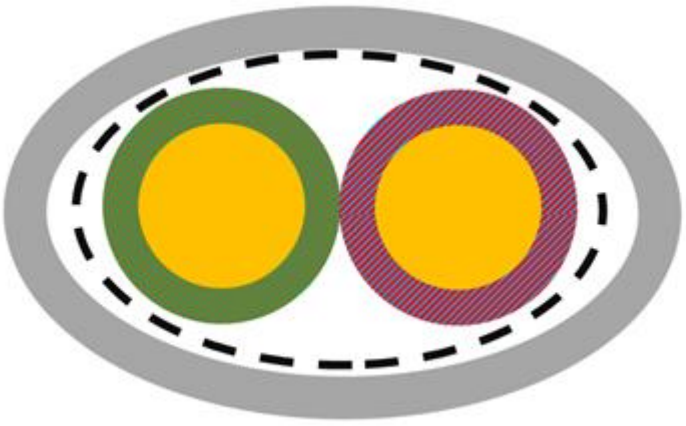


Figure 11: Multi Core Illustration

This is an illustration for the multi core example that describes the specification of a multi core wire. The example uses a multi core, which is insulated with a grey insulation, shielded and which contains two FLRY cores of different colouring that are a twisted pair.

3.4.3 Multi Core Specification

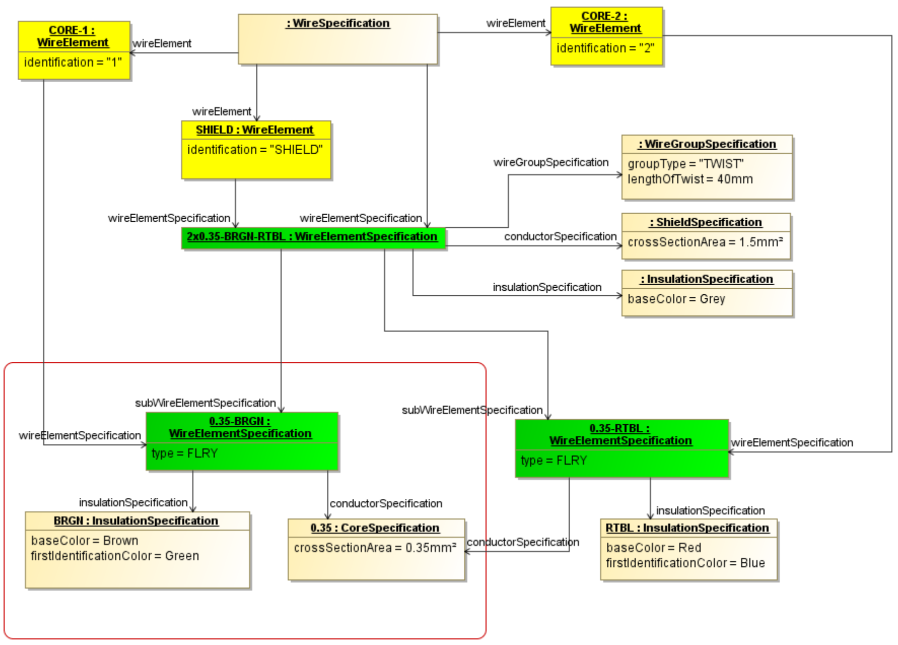


Figure 12: Multi Core Specification

The Figure displays the instantiation of the multi core wire example in the VEC. The hierarchy of the wire is highlighted in the figure in green. The specification of the brown and green FLRY core (0.35-BRGN) is actually the same as the specification in the single core example (highlighted by the red outline in the figure). It is reused and not defined redundantly. Since it is the same WireElementSpecification object, the context specific naming of the WireElement is necessary, as mentioned in single core example (highlighted in yellow).

The only difference between the two WireElements representing the two cores is the coloring of the insulation. Therefore the two WireElementSpecifcations share the same CoreSpecfication, but have individual InsulationSpecifications. In the context of the displayed WireSpecification the Brown & Green Core is addressed with the identification “1”, the second Core is addressed with the identification “2”. The two Cores are grouped together by a third WireElementSpecification (2x0.35-BRGN-RTBL).

This third WireElementSpecification defines the type of grouping by a WireGroupSpecification. In the example the Grouping is the definition of a twist of the two Cores. It also defines the insulation around the two SubWireElements by an InsulationSpecification. Since the defined wire has a conductive shield as well, the described WireElementSpecification references a ShieldSpecification, too. The cross section area in the ShieldSpecification defines the nominal cross section area of the conductive material used in the shield.

3.5 *Connectors*

3.5.1 Modular Connector

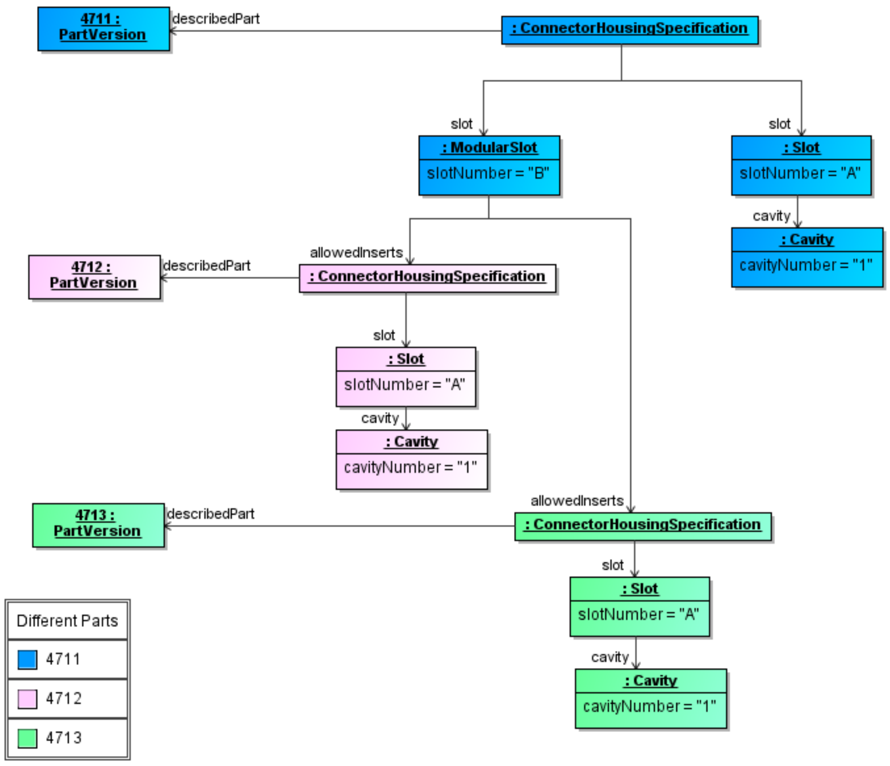


Figure 13: Modular Connector

This tutorial illustrates the definition / description of modular connectors. A modular connector is a connector that can be recursively assembled, so that certain fields can fitted with different other parts.

In the displayed example the *PartVersion* "4711" is a modular connector. The *ConnectorHousingSpecification* defines a regular *Slot* "A" with a number of cavities and a *ModularSlot* "B". This *ModularSlot* is compatible to two different inserts (defined by individual *ConnectorHousingSpecifications*). The two *PartVersion* "4712" & "4713" define these allow inserts.

3.5.2 Modular Connector Instancing

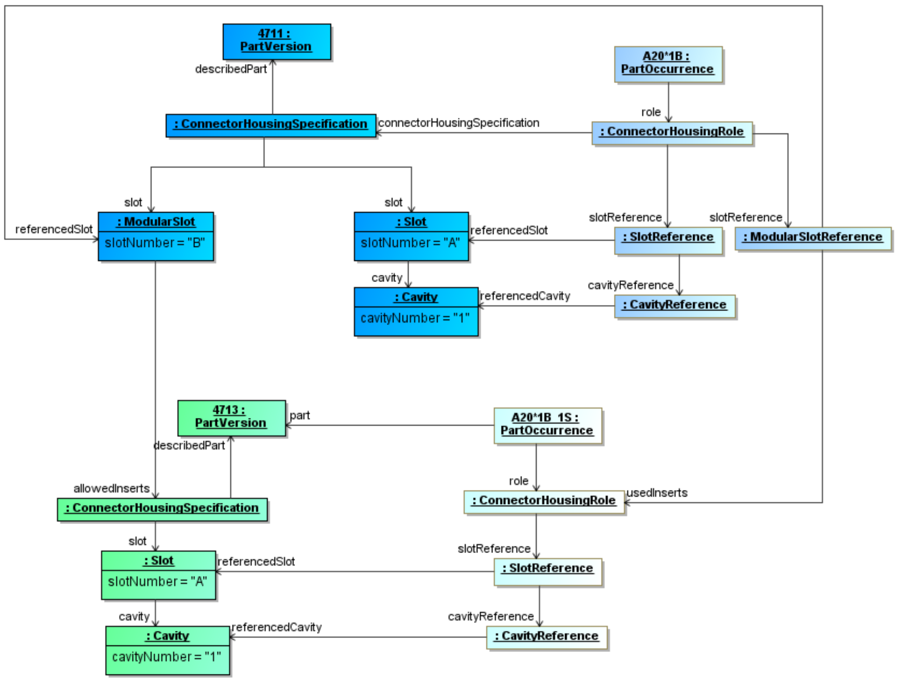


Figure 14: Modular Connector Instancing

The diagram shows the instantiation of modular connector (previous example). On the left hand side of the diagram the component description of the modular connector is shown (similar to the previous example). On the right hand side the instancing of such a modular connector is shown.

Both parts of the modular connector (the housing and the insert) have their own *PartOccurrence*. The *ModularSlotReference* defines which inserts are actually used in the specific context.

Note: As a wiring harness is often described in a 150% scope, it is possible that a *ModularSlotReference* references more than one *ConnectorHousingRole* as *usedInserts*. In these cases the variant management mechanisms have to ensure, that in a concrete case only one insert is used. This can be either done explicitly with *PartStructureSpecifications* or implicitly with a *VariantConfiguration*.

3.5.3 Connector with multiple Segment Connection Points

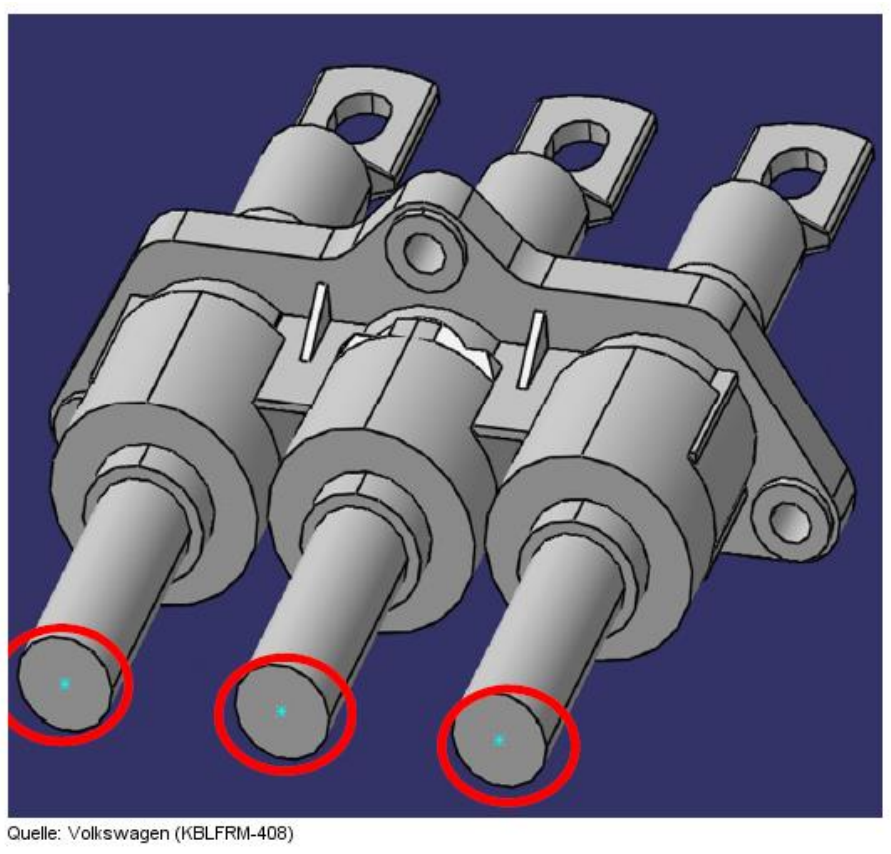


Figure 15: Connector with multiple Segment Connection Points

3.5.4 Instancing

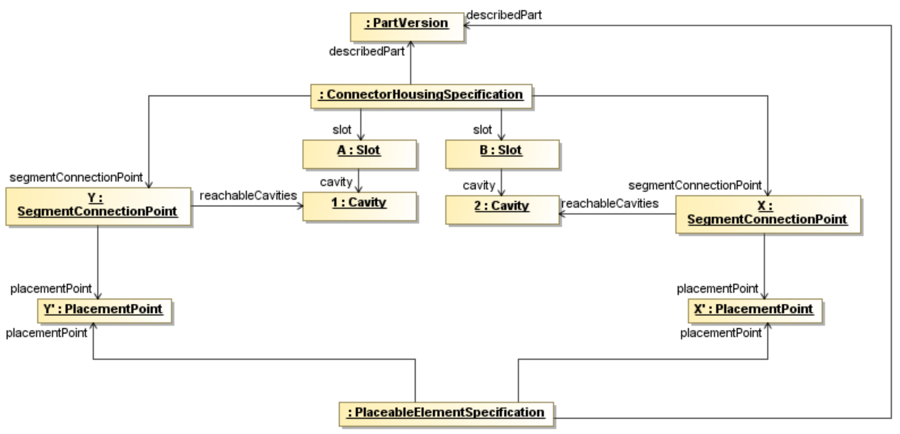
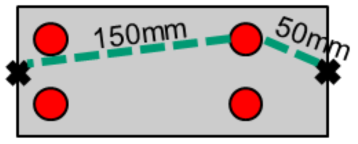


Figure 16: Instancing

The example shows a connector that has two cavity, that are only reachable through different segment connection points. By associating these *SegmentConnectionPoints* with corresponding *PlacementPoints* the *SegmentConnectionPoint* become 'placeable' on nodes in the topology of a harness.

3.5.5 Addons for Cavities



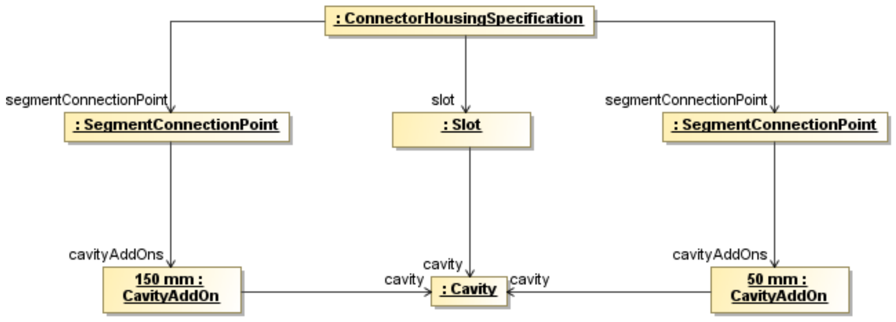
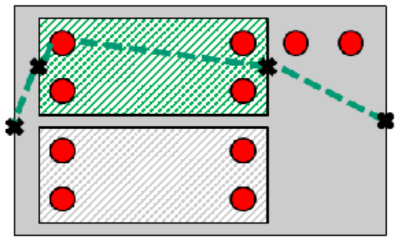


Figure 17: Addons for Cavities

This example shows how add-ons for cavities in a connector could be defined. In this example, the *ConnectorHousingSpecification* has two different *SegmentConnectionPoints.* Each of them is defining it's own *CavityAddOn.* So depending on the *SegmentConnectionPoint* used, a *Cavity* can have for example 50mm as well as 150mm as Add-On.

3.5.6 Addons for Modular Slots



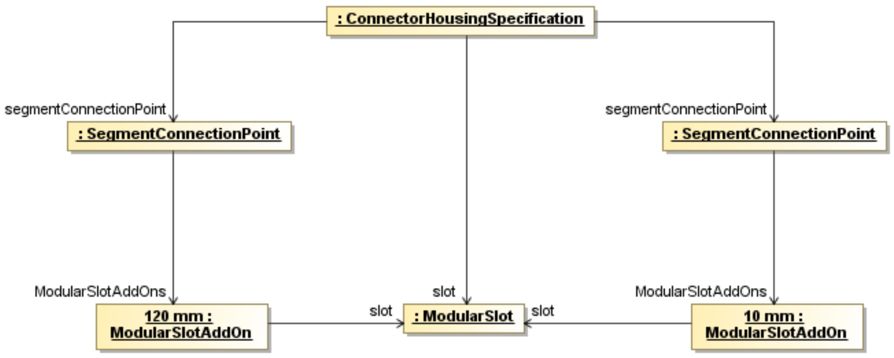


Figure 18: Addons for Modular Slots

If a *ConnectorHousingSpecification* has *ModularSlots*, the Add-ons are not defined individually for all cavities for all possible inserts, but only per *ModularSlot*. The Add-On defined in the *ModularSlotAddOn*, is the Add-On need to reach the *ModularSlot* from the corresponding *SegmentConnectionPoint*. The add-on needed to reach a certain cavity in an used insert, can be obtain from *ConnectorHousingSpecification* of the used insert.

3.5.7 Simple ConnectorHousingCap wireAddOn

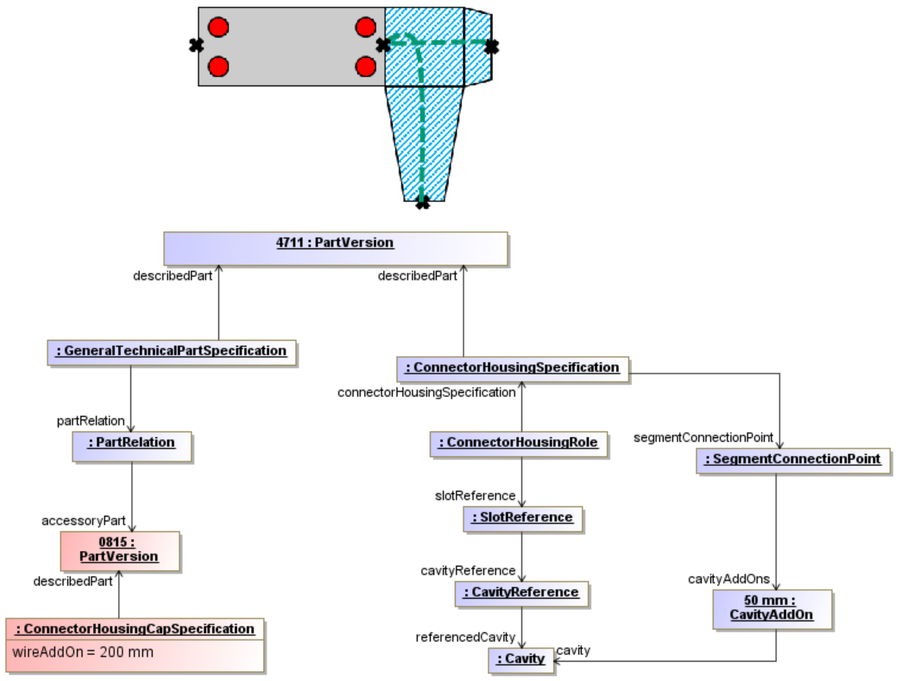


Figure 19: Simple ConnectorHousingCap wireAddOn

Wire add-ons caused by cap's are defined in the*ConnectorHousingCapSpecification*. The specified value is the add-on required to reach the *SegmentConnectionPoint* of the ConnectorHousing from the entry point of the cap.

3.6 ECUs and EE Components

3.6.1 Simple EE Component

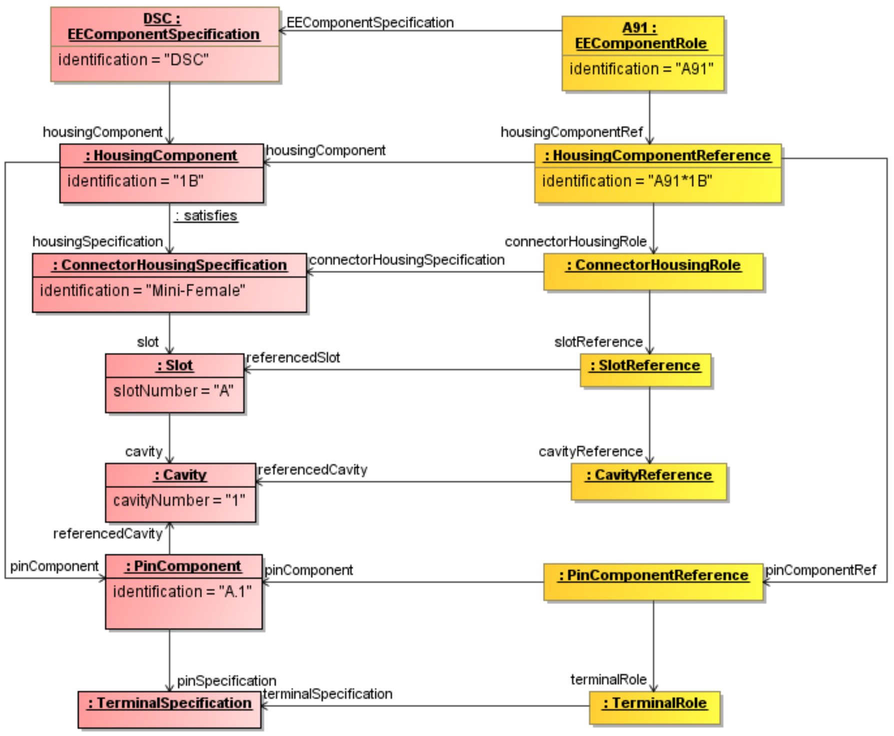


Figure 20: Simple EE Component

The diagram illustrates a complete specification and instantiation of a EEComponent. The master data definition is shown on the left hand side (highlighted in red), the instantiation on the right hand side (highlighted in yellow).

An EE-Component is defined by an *EEComponentSpecification*. An EE-Component can have multiple *HousingComponents*, which can be defined more detailed with a *ConnectorHousingSpecification.* These *HousingComponents* have *PinComponents* representing the different electrical pins of an EE-Component. The *PinComponent* can carry specific properties of the Pin (e.g. the Logical Signal). The physical properties of the pin can be defined by an associated *TerminalSpecification* (e.g. the maximum current for the pin). A *HousingComponent* can have multiple pins with different properties sharing the same *TerminalSpecification*.

When the EE-Component is used, it must be instantiated. For each subelement of the *EEComponentSpecification* a corresponding element is used for the instantiation. The doubled instantiation of the pin with a *PinComponentReference* and a *TerminalRole* is needed, in order to provide a common interface for other areas of the VEC. For example the Mating defines the connection between the sides of an inliner or between a harness connector and an ECU by using the *TerminalRole*.

3.6.2 Switching States Illustration

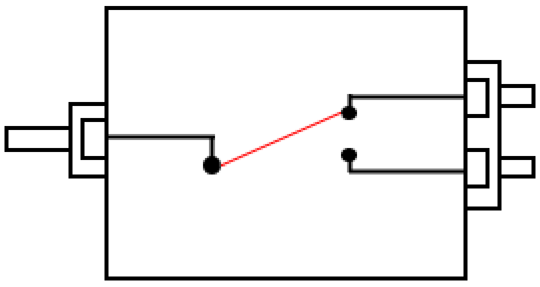


Figure 21: Switching States Illustration

This is an illustration for an *EEComponent* with *SwitchingStates*. It displays a simple switch with two states. The example uses a *EEComponent* with two slots. The first slot has one cavity with one pin component. The second slot contains two cavities with two pin components. The pin in the first slot is connected to the pins in the second slot with switchable connections.

3.6.3 Switching States

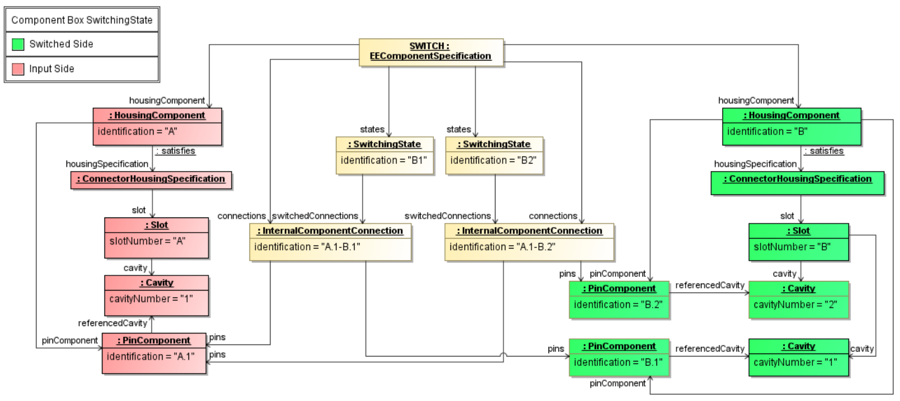


Figure 22: Switching States

The figure displays the specification of a *EEComponent* with switchable internal connections (*SwitchingStates*). The *EEComponent* is illustrated in the previous diagram.

The input side of the *EEComponent* is highlighted in red and the output side is highlighted in green. The component box has two *HousingComponents* specified by a corresponding *ConnectorHousingSpecification*. The input side *HousingComponents* has one slot with slot number “A” containing one cavity with cavity number 1 and one *PinComponent* with identification A.1 referencing the cavity. The output side *HousingComponent* has one slot with slot number B containing two cavities with cavity number 1 and 2. Each cavity is referenced by a corresponding *PinComponent* with identification B.1 respectively B.2.

The internal connections between the *PinComponents* on the input and output side are described by two *InternalComponentConnections*. Which *InternalComponentConnection* are available in a defined state is expressed by the two *SwitchingStates* B1 and B2. If the *EEComponent* is in state B1, the

InternalComponentConnection A.1-B.1 is available. If the *EEComponent* is in state B2, the *InternalComponentConnection* A.1-B.2 is available.

3.6.4 Component Assignment for Component Boxes

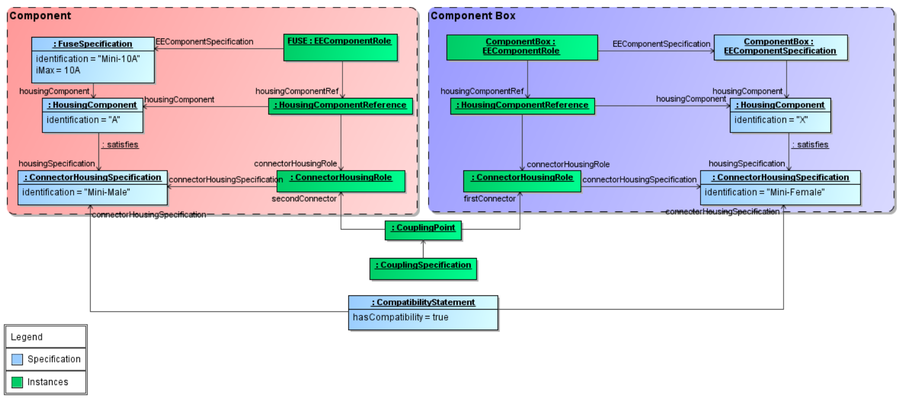


Figure 23: Component Assignment for Component Boxes

The diagram illustrates the most basic assignment of components (e.g. a fuse) to a component box. On the left side of the diagram a *Component* is shown, on the right side a *ComponentBox.* The specifications (part master data) are highlighted in blue, the instances are highlighted in green.

Regular *Components* (e.g. Fuses) have a single *HousingComponent* with a *ConnectorHousingSpecification*. The *ConnectorHousingSpecification* defines the geometric properties the *Component.* The compatibility between to *ConnectorHousingSpecifications* is expressed with a *CompatibilityStatement*.

The assignment of a Component in its slot in a *ComponentBox* for a specific instance is done the standard mechanism in the VEC for the connection of Components, Connectors etc., the *CouplingPoint*. In its most basic variant, only the assignment between two *ConnectorHousingRoles* is defined. For more detailed definition see the *CouplingPoint* examples.

3.7 Pinning

The following section contains examples for the definition of Pinning information in the VEC. This means the specification of the electrical behavior of *PinComponents*.

3.7.1 Signal Peak Distance and Duration

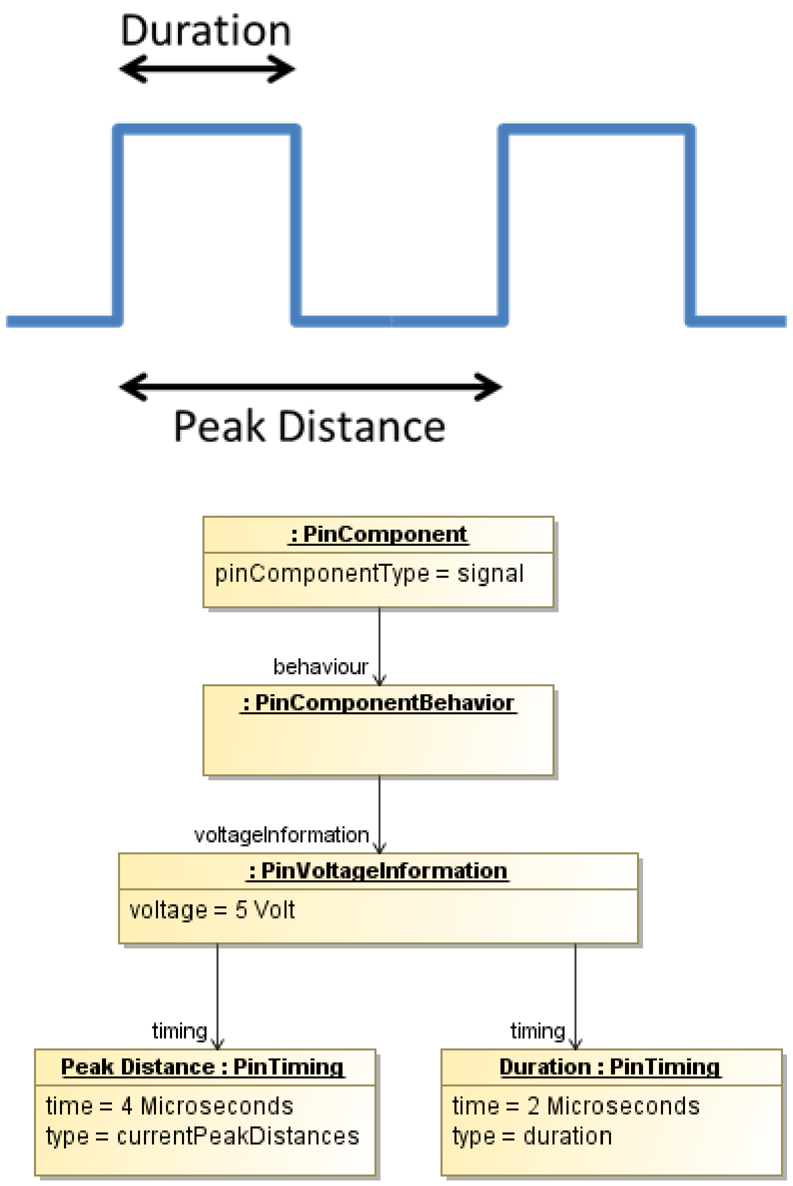


Figure 24: Signal Peak Distance and Duration

This example shows how a digital signal with pulse width and pulse separation can be defined. The *PinComponentBehavior* of a *PinComponent* has *PinVoltageInformation*. The *PinVoltageInformation* can define multiple *PinTiming* definitions. For pulsed digital signals, two *PinTimings* are used. One *PinTiming* (Duration) describes the pulse width. The other *PinTiming* (Peak Distance) describes pulse separation.

3.7.2 Load Type Dependant Maximum Current (Relais)

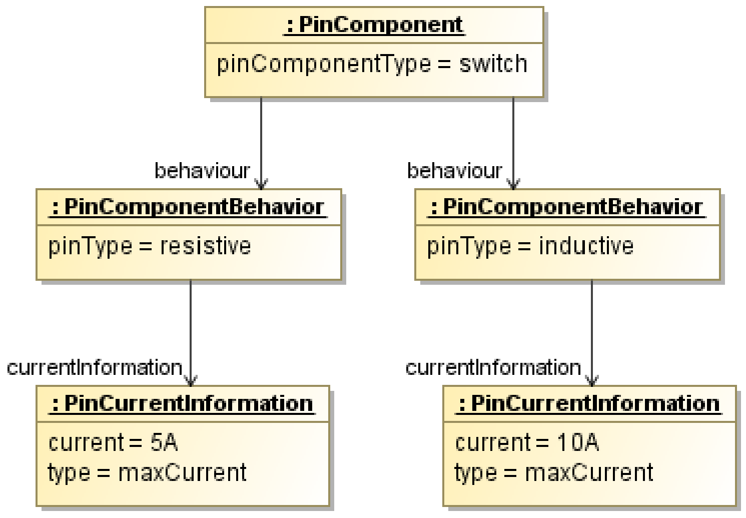


Figure 25: Load Type Dependant Maximum Current (Relais)

Dependant on the load type (inductive, resistive, capacitive) a switching contact of a relais can have different maximum loads.

The diagram shows a *PinComponent* of type switch that has two *PinComponentBehaviors* with pinType resistive and inductive. Each *PinComponentBehavior* has a *PinCurrentInformation* with type maxCurrent and different current values.

3.8 Accessories

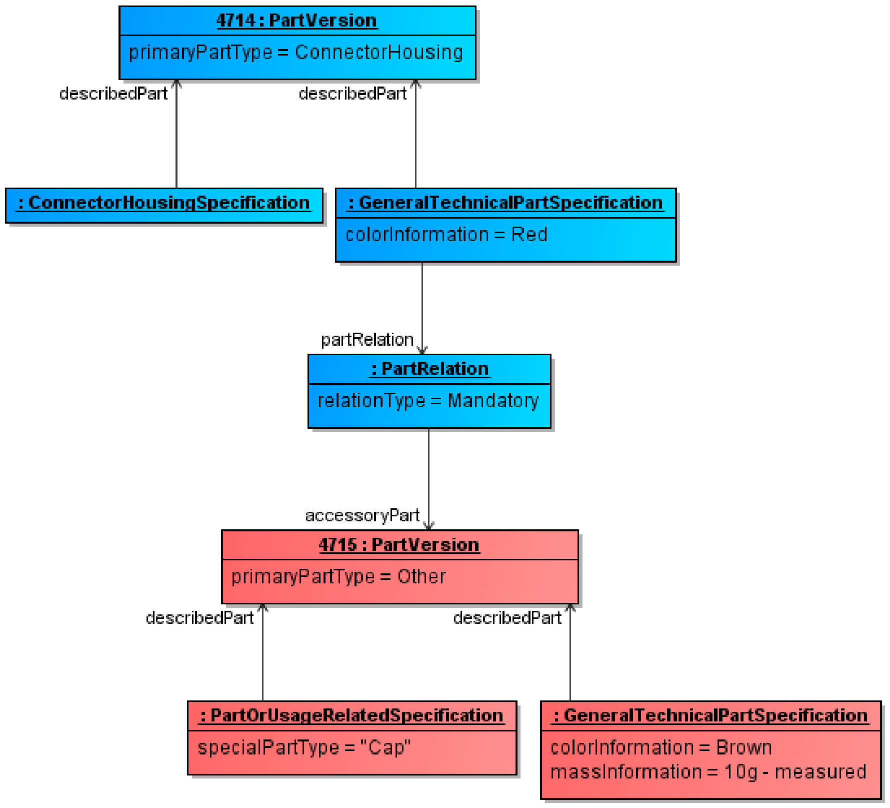


Figure 26: Accessories

Accessories are parts that are not further specified in the VEC (e.g. caps or locks for connectors). Those components are marked with the *primaryPartType="Other"* and a *PartOrUsageRelatedSpecification* that can be used to define which type of "accessory" it is. Common part attributes can be defined with a *GeneralTechnicalPartSpecification*.

A relation between *PartVersion* and its accessories can be established with a *PartRelation* in a *GeneralTechnicalPartSpecification*.

3.9 Grommets with Placement

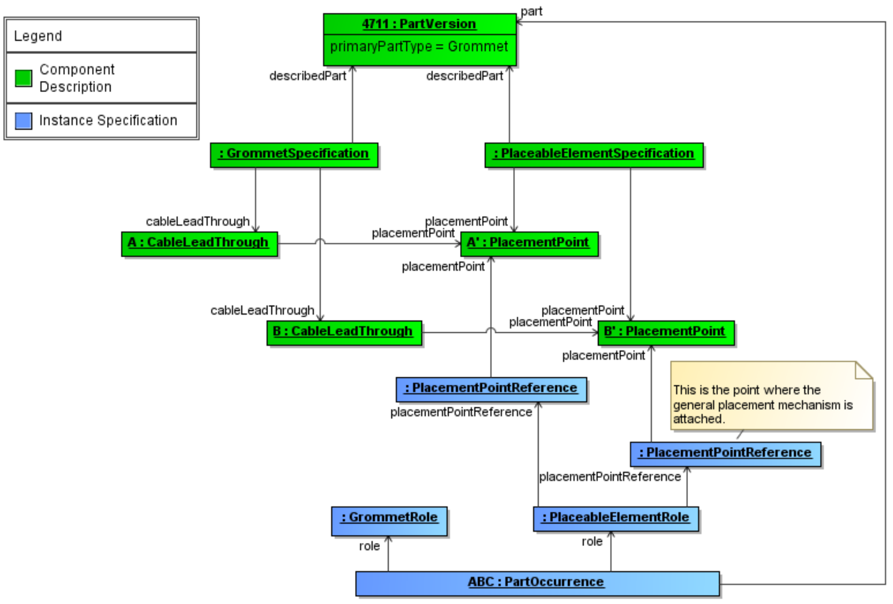


Figure 27: Grommets with Placement

Grommets have certain points (the *CableLeadThrough*) that are relevant for the placement of a grommet. Since currently the *CableLeadThrough* only specifies relevant information on the component side and not on the instance side, there is no corresponding subelement of the *GrommetRole*. The ability to place a *CableLeadThrough* on a specific point in the topology is covered by the generic mechanism of *PlacementPoints* & *PlacementPointReferences.*

3.10 Channels

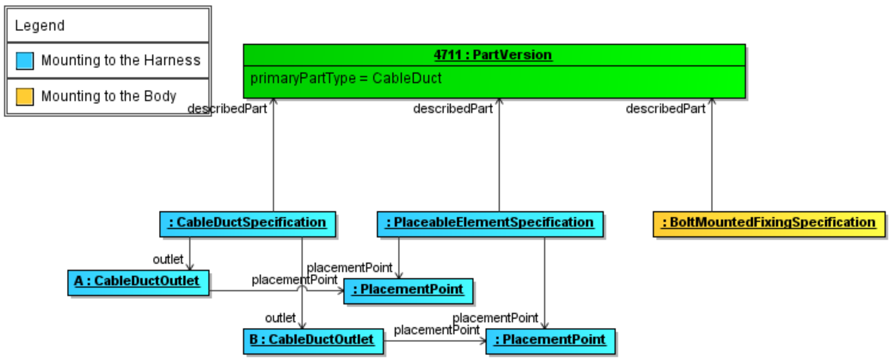


Figure 28: Channels

CableDucts can fulfill have to relevant aspects. They can be connected with the harness (placement). This is specified with the blue classes. At the same time a *CableDuct* can have properties of a *Fixing* since it is often mounted to the vehicle body as well. This aspect is defined by an additional *FixingSpecification*.

3.11 Fixings

3.11.1 Fixing with PlacementPoints Illustration

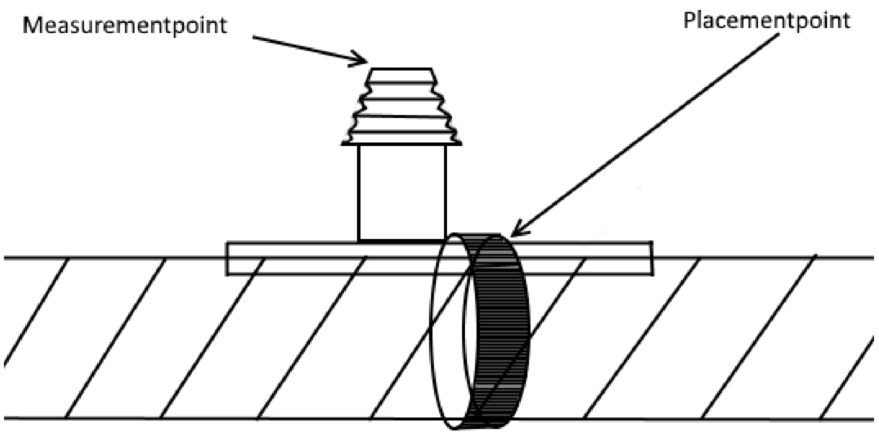


Figure 29: Fixing with PlacementPoints Illustration

This illustration shows the *Fixing* with a *PlacementPoint* and *MeasurementPoint* as a PartOccurrance.

3.11.2 Fixings with PlacementPoints

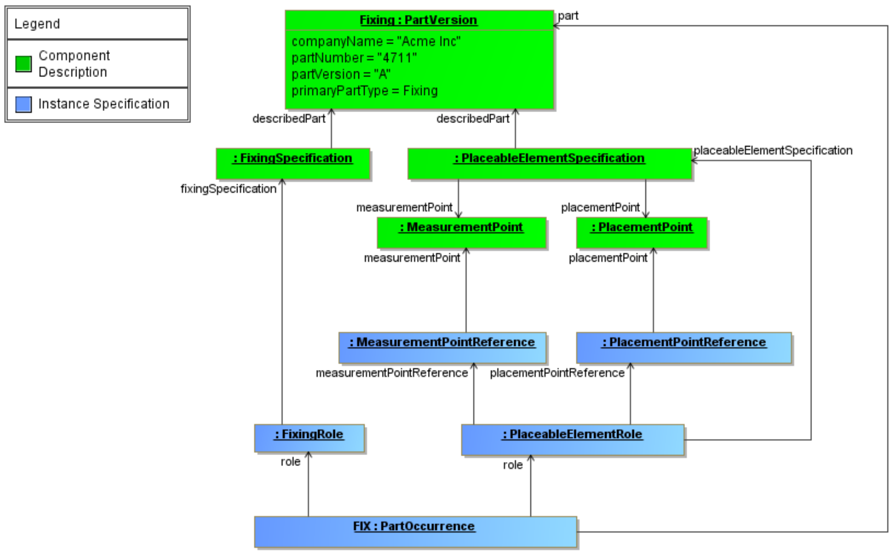


Figure 30: Fixings with PlacementPoints

The ability to place a *Fixing* on a specific point in the topology is similar to Grommets covered by generic mechanism of *PlacementPoint* and *PlacementPointReference*. Additionally the measurement of *Fixing* is covered by *MeasurementPoint* and *MeasurementPointReference*.

3.11.3 Fixings with additional Cable Ties

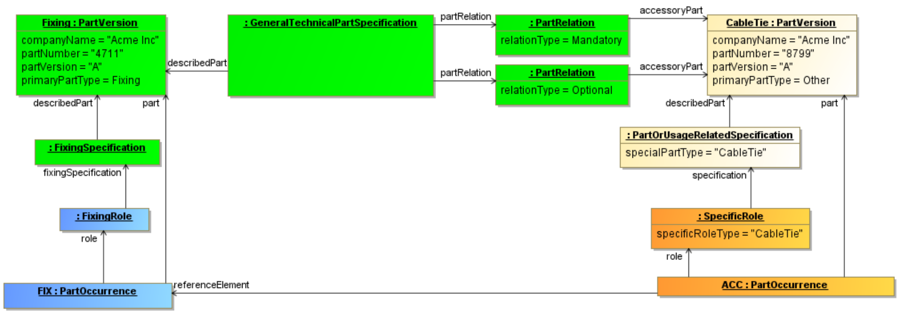


Figure 31: Fixings with additional Cable Ties

The diagram illustrates the definition of a *Fixing* with *CableTies* as *Accessory*. The upper half of the diagram is the definition of the part master data.

The Fixing is described with a *PartVersion* and a *FixingSpecification.* To describe its accessories it has *GeneralTechnicalPartSpecification* with *PartRelations* to link the accessories. In this case, one *CableTie* is mandatory, a second one is an optional add on. Both are referencing the *PartVersion* of the *CableTie*.

The *CableTie* is currently defined with a not further detailed *PartOrUsageRelatedSpecification,* since there is no *CableTieSpecification* in the VEC at the moment. That the accessory is a *CableTie* is defined by the value of the *specialPartType* attribute.

If there are any additional properties necessary for the *CableTie*, then they could be specified with *CustomProperties* for the *PartOrUsageRelatedSpecification.*

For the instancing of these components, both are created with a *PartOccurrence.* The *Fixing* is defined with a *FixingRole*, the *CableTie* with a *SpecificRole* (For the same reasons why a *PartOrUsageRelatedSpecification* has been used).

4 Topology and Geometry

4.1 Placements

A Placement defines the way how a component is associated to the topology. The following sections contain examples about the different types of placements.

4.1.1 Simple WireProtection Illustration

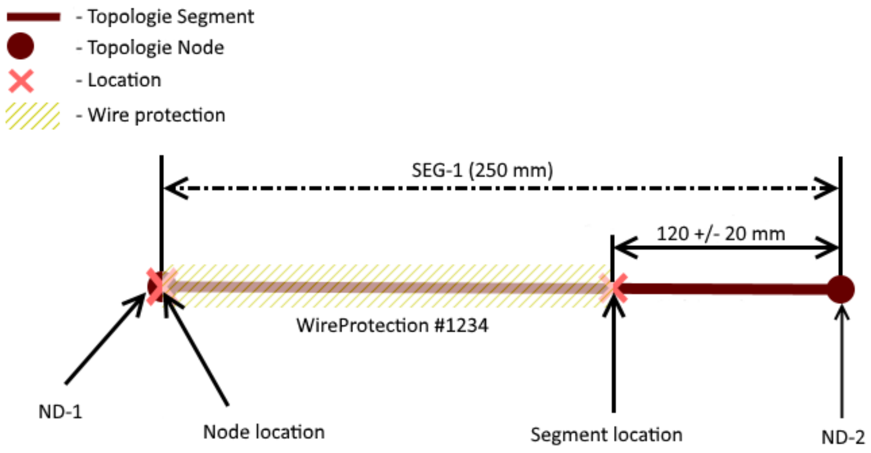


Figure 32: Simple WireProtection Illustration

This diagram illustrates the placement of a simple *WireProtection* as shown in next diagram.

4.1.2 Simple WireProtection

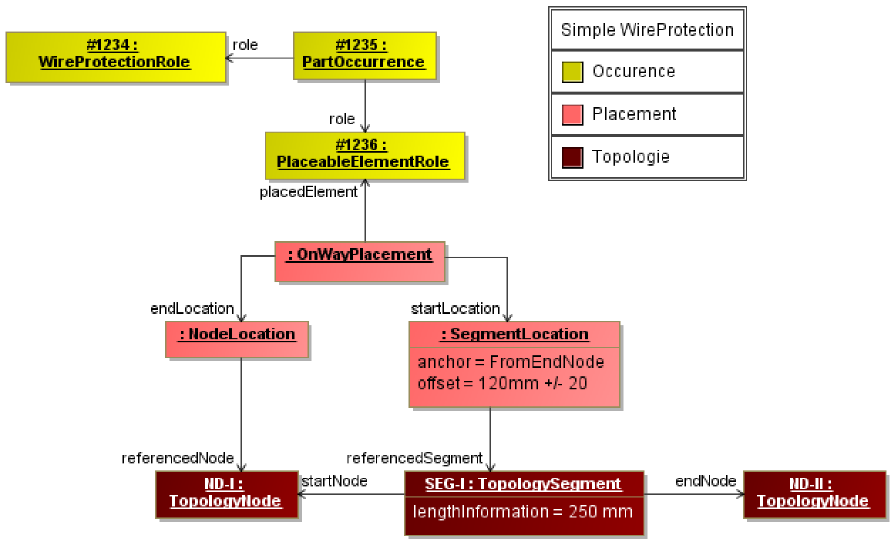


Figure 33: Simple WireProtection

The Figure above displays the placement of a simple wire protection. The *PartOccurrence* is placed with an *OnWayPlacement* via a *PlacableElementRole*. This means the placed component covers a linear area of the harness topology. The start and the end of this area is defined with two *Locations*. In the shown situation the *StartLocation* is a *SegmentLocation*, which means the start is somewhere in the middle of a *TopologySegment*. It is defined to be at 120mm measured from the *EndNode* of the *TopologySegment*. The *EndLocation* of the WireProtection is located on a *TopologyNode* with a *NodeLocation*. It is not valid to define *Location* with *SegmentLocation*, which could be also expressed by *NodeLocations*. This means for *SegmentLocations* an offset of 0 or equal to the segment length is illegal.

Since the offset is *NumericalValue* it can have an optional *Tolerance.*

4.1.3 WireProtection with Dimension Illustration

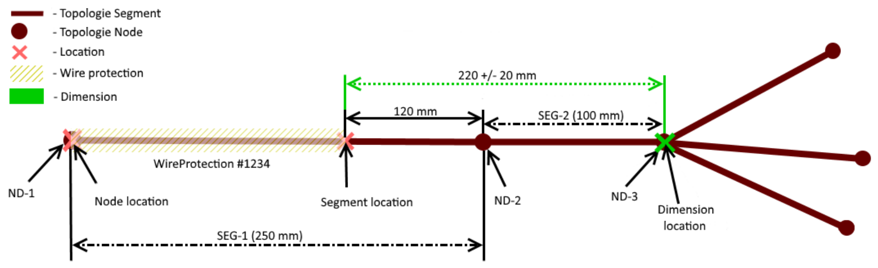


Figure 34: WireProtection with Dimension Illustration

This diagram shows the previous example extended with a *Dimension.* In the previous example, the beginning of the *WireProtection* was defined with a tolerance value. This method is applied, if the tolerance is applied to the next *TopologyNode* (Start- or End-Node of the segment).

In many cases, tolerances are defined relative to specific points (e.g. points that can be measured) somewhere in the topology. In these cases a *Dimension* is used to defined the tolerance.

4.1.4 WireProtection with Dimension

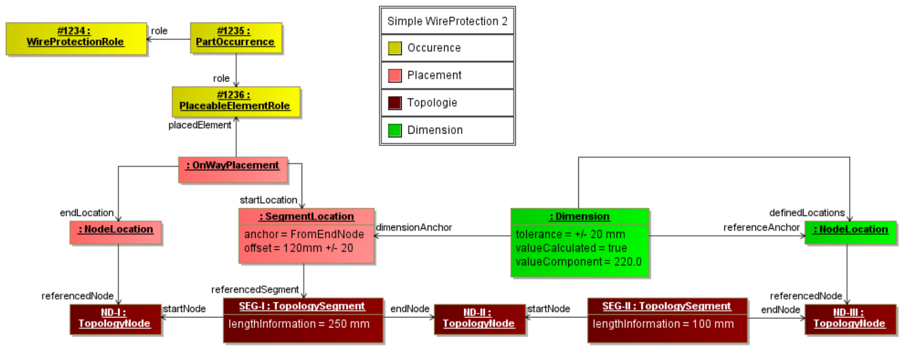


Figure 35: WireProtection with Dimension

The placement of the *WireProtection* is just the same as in the previous example. It is extended with the *Dimension* (highlighted in green). The *Dimension* defines the tolerance of +/- 20mm between the *TopologyNode* ND-III and the beginning of the *WireProtection*.

The fact, that the *Dimension* is specified between the *TopologyNode* and the beginning of the *WireProtection* is expressed, that the *TopologyNode* is referenced directly (with a *NodeLocation* contained by the *Dimension*). The *SegmentLocation* used as *DimensionAnchor* is the same as used for the placement of the *WireProtection*.

The *valueCalculated=true* flag of the *Dimension* indicates that the *valueComponent* (220mm) is a derived an calculated value and not a user defined value. This value can be obtained from the placement information and the lengths of the *TopologySegments*.

4.1.5 Fixing Placement Illustration

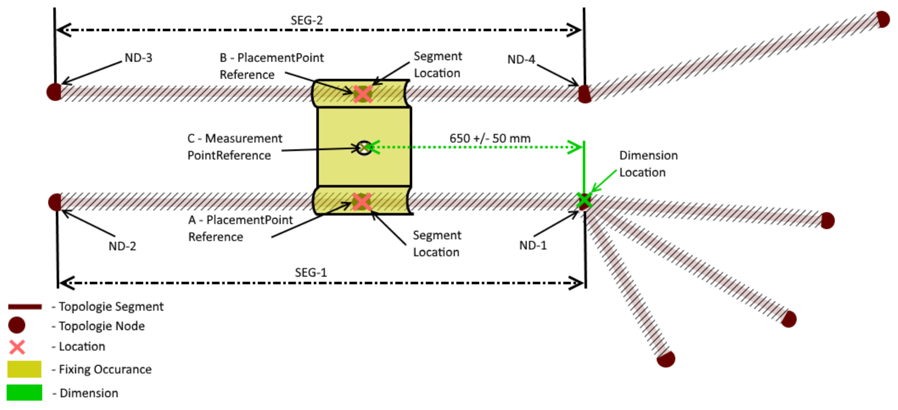


Figure 36: Fixing Placement Illustration

This diagram illustrates a more complex placement situation, including the usage of dimension.

The illustration shows a bracket, that is placed independently on two *Segments* (SEG-1 & SEG-2). The two points where the bracket is placed on the *Segments* are identified separately (*PlacementPointReference A & B)*. Additionally a *Dimension* is added, which gives a *Tolerance* between a geometric point (e.g. a bolt) on the bracket (*MeasurementPointReference C*) and a *Node* (ND-1) in the Topology.

4.1.6 Fixing Placement

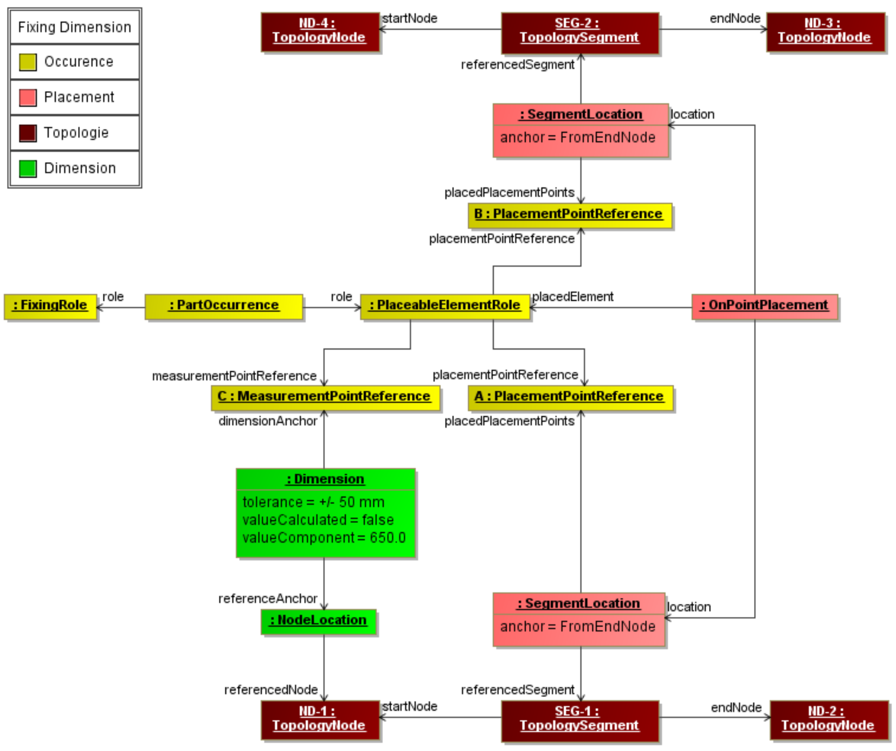


Figure 37: Fixing Placement

The diagram illustrates the instantiation of the example in the preceding diagram. Since the *PartOccurrence* can be placed in the topology, it has a *PlaceableElementRole* (with a corresponding *PlaceableElementSpecification* not shown in the diagram). The points where it can be placed onto the topology are represented by the *PlacementPointReferences A & B*. The point which can be used as anchor for a dimension (which can be any reference point on the component), is represented by the *MeasurementPointReference C.*

The actual placement is done with an *OnPointPlacement* which has two *SegmentLocations.* One for each *PlacementPointReference.*

4.1.7 Large Area WireProtections

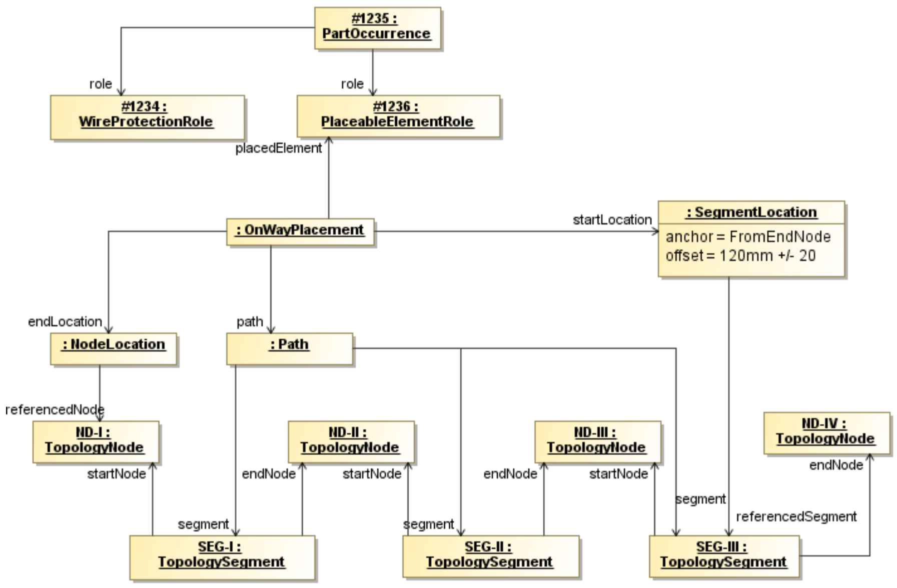


Figure 38: Large Area WireProtections

In some cases it is necessary to place a wire protection over a greater area of the topology, consisting of more than one TopologySegment (e.g. Tubes with a fixed length). For these cases the OnWayPlacement defines two locations for the start and the end and a path along which the wire protection is placed. The path is an ordered list of the segments from the start to the end. If a SegmentLocation is used for the start or the end the path must contain these segments as well.

For each TopologySegment the use of Start- and End-Node has no semantik relevance. The names are just used to make it possible to identify the corresponding TopologyNodes correctly e.g. when defining the anchor for a SegmentLocation.

4.1.8 Fixed Components (Single Location)

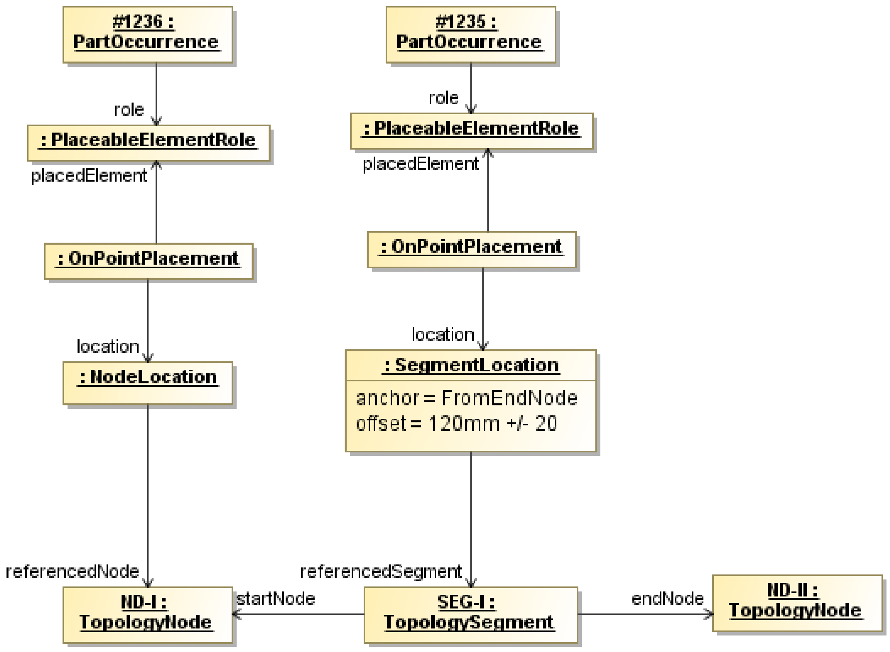


Figure 39: Fixed Components (Single Location)

Fixed components are elements that are placed on a certain point in the topology, such as Connectors, Fixings and so on. These components are placed with an OnPointPlacement as shown in the example. If the Component has to be placed on a Node (e.g. a Connector) a NodeLocation is used. If the Component has to be placed on a Segment a SegmentLocation is used. The usage and constraints for the Locations are the same like the ones for OnWayPlacements.

4.1.9 Fixed Components (Multiple Locations)

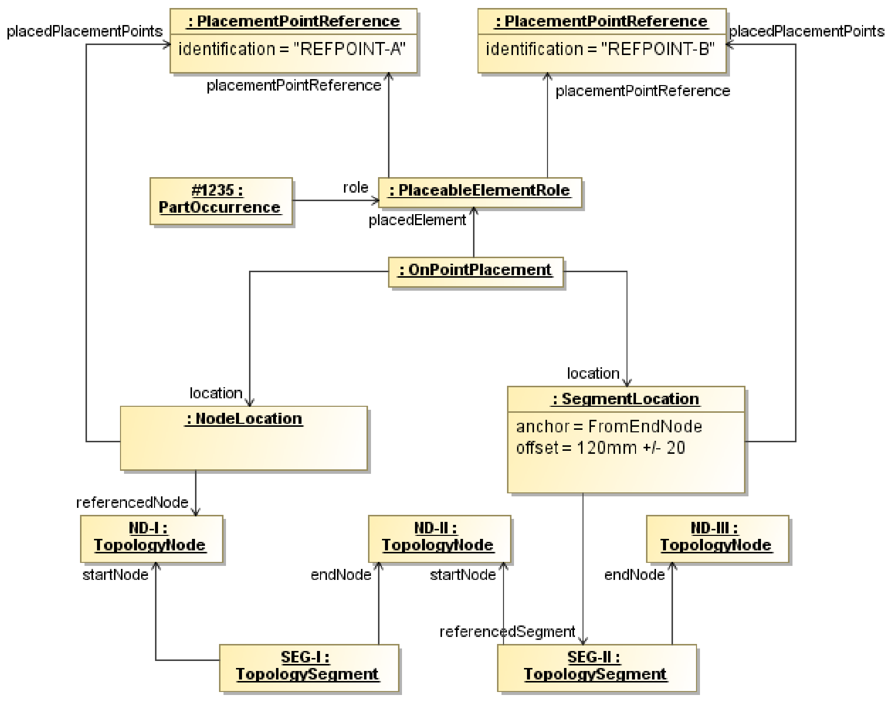
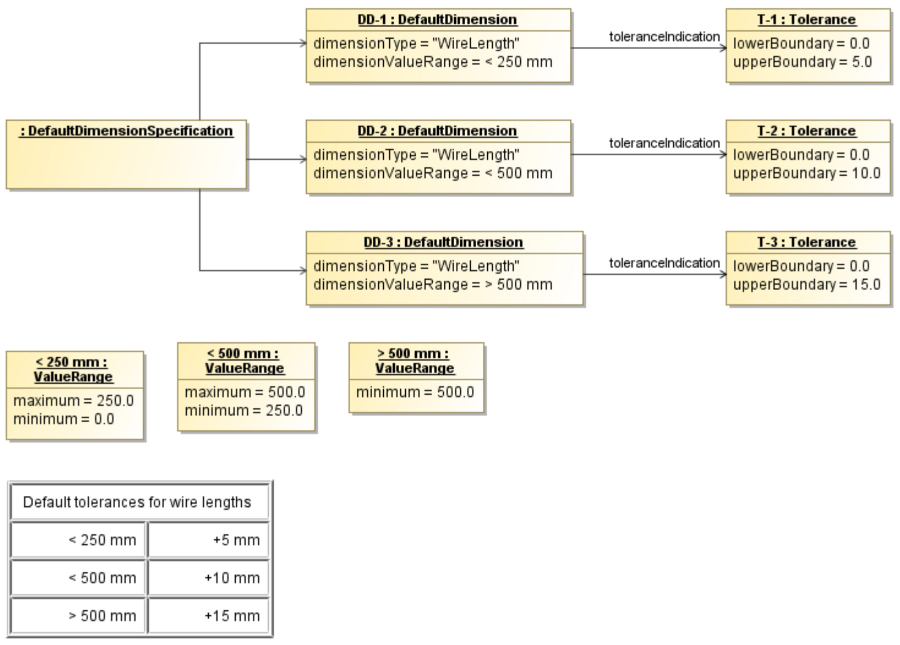


Figure 40: Fixed Components (Multiple Locations)

Some components, for example channels or a large connector with more than one segment connection point, may be placed on multiple positions in the Topology. For example a channel can have two or more reference points (e.g. the outlets) that must be associated to the different positions topology. In these cases an OnPointPlacement with more than one location is used. In order to identify which location places which point of the component (e.g. the outlets), a PlaceableElementRole can define PlacementPointReferences which are creating a relationship to the component description.

4.1.10 Default Dimensions



Default tolerances for wire lengths	
< 250 mm	+5 mm
< 500 mm	+10 mm
> 500 mm	+15 mm

Figure 41: Default Dimensions

The diagram illustrates the use of a *DefaultDimensionSpecification.* The *DefaultDimensionSpecification* can be used to specify default dimensions / tolerances for certain attributes and *ValueRanges.* In this examples the *Specification* is used for the length of wires. (indicated by the *dimensionType)*. The *dimensionValueRange* defines for which value's of this type, the referenced *Tolerance* is applicable.

In this example for a wire length lower than 250 mm a *Tolerance* of +5 mm is allowed, for values between 250 mm and 500 mm a *Tolerance* of +10 mm is allowed and for everything above 500 mm a *Tolerance* of 15 mm is allowed.

5 Connectivity

5.1 Contacting Specification

The contacting in the VEC defines the relationship between Cavities, WireEnds, Terminals and Seals. Since there are various types of contactings possible, the different types are not defined explicitly in the VEC. The VEC offers a quite generic structure (the Contacting), which should be able to support all the different possible types. This is necessary, because the different contacting types are driven by technical requirements and new contactings might emerge over the time. The downside of the generic structure is that the structural schema allows constellations that are not sensible from a technical point of view as well. The following sections show the different contacting types used today, and how they have to be implemented in the VEC.

Since the contacting can be used for different levels of abstraction (Product Definiton or Electrological Wiring) only the “Role-Side” of the necessary objects is shown. Necessary *PartOrUsageRelatedSpecifications*, *PartOccurrences*, *PartUsages* and *PartVersion* are omitted.

5.1.1 Standard Contact

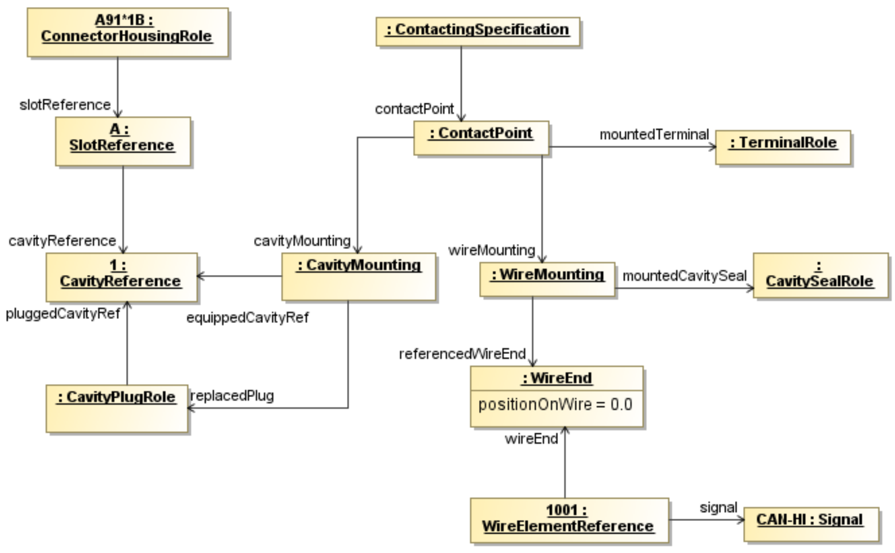


Figure 42: Standard Contact

A standard Contacting is the most common case in a wiring harness. For a standard contact you have one wire end that has one terminal crimped on it, which is placed in one cavity. It exists in two variants, sealed and unsealed. The example displays the sealed variant, for the unsealed variant the CavitySeal and the CavityPlug have to be omitted.

The contacting is defined by a ContactPoint, contained in a ContactingSpecification. It is possible (and recommended) to define multiple ContactPoints in one ContactingSpecification. Usally there exists one ContactingSpecification per DocumentVersion in the scope. So for example if the VEC-File represents a 150%-Harness definition, then you will have on ContactingSpecification for the complete harness.

A ContactPoint is defined as a point of exactly one electrical potential, which means all conducting components related to the ContactPoint are short-circuited.

The ContactPoint defines the terminal that is used for the contacting. It is then split up into two parts, the side of the crimp of the terminal (represented by the WireMounting) and the side of the terminal, which is placed in the cavity (represented by the CavityMounting). Since the example is about a sealed standard contact, the WireMounting displayed references exactly one CavitySeal and one WireEnd, which means these two components are crimped together onto the terminal. On the other side the CavityMounting defines the Cavity in which the terminal will be placed. For a sealed environment it is necessary, that the Cavity is plugged with a CavityPlug, in case the Cavity is not occupied by a contacting. If the Cavity is occupied, the CavityMounting defines explicitly, which CavityPlugs are replaced by its existence.

5.1.2 Multi Crimp Contact

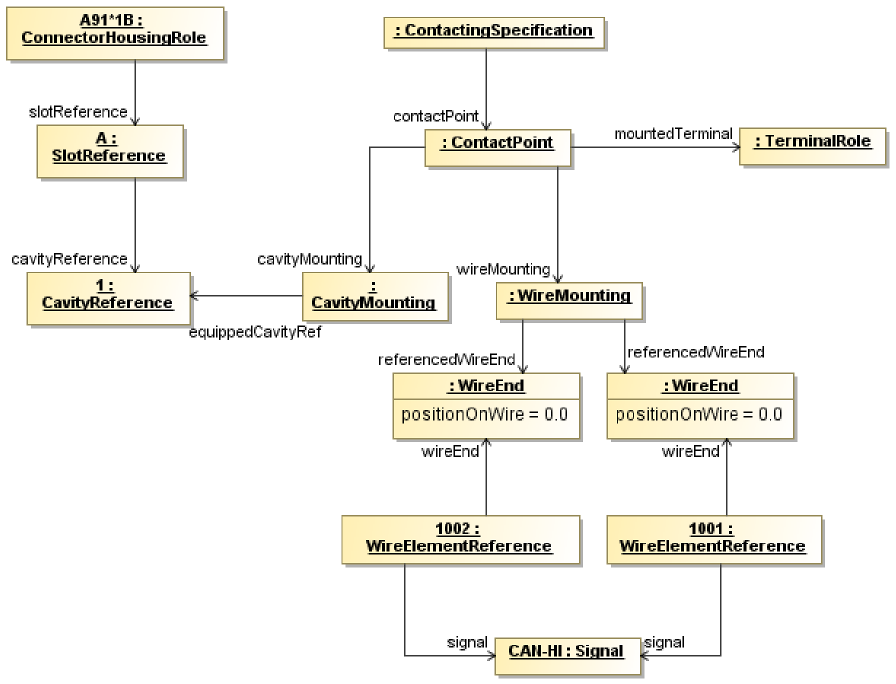


Figure 43: Multi Crimp Contact

A Multicrimp is quite similar to the standard contact, with the difference that there is more than one wire crimped onto the terminal. Therefore the displayed example is quite the same. The differences are:

- There is no CavityPlug or CavitySeal, since it is not usefull / possible from a technical point of view to seal a multicrimp.
- There are two (or more) WireEnds associated with the WireMounting.

For clarification of the example the two WireElementReferences reference their Signal. It is the same, since the two WireEnds are crimped onto one Terminal and therefore they are set to one single electrical potential. This is only displayed in the example in order to make it clear, what is meant by “A ContactPoint has one single electrical potential”. This is not a restriction of the VEC, since the development processes might need (or create) different signal names for the same electrical potential.

5.1.3 Ringterminal - Splice

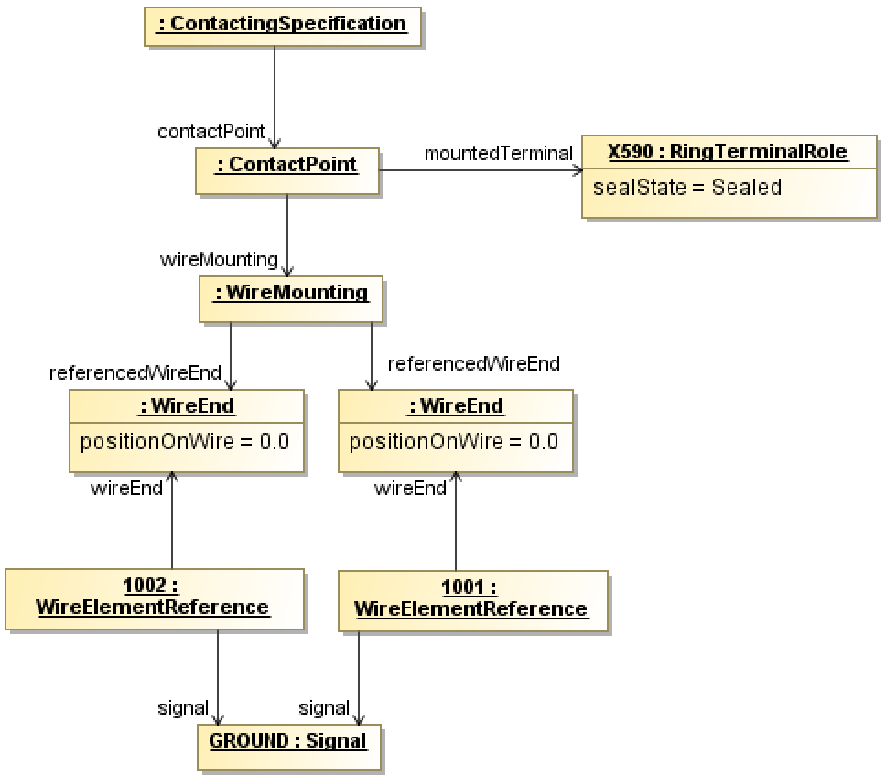


Figure 44: Ringterminal - Splice

The structure displayed in the example applies to ring terminals and splices as well. On the side of the wire it is the same as a multi crimp. The difference is that no cavity mounting is used, since a ring terminal / splice has no cavities.

5.1.4 Bridge Terminal

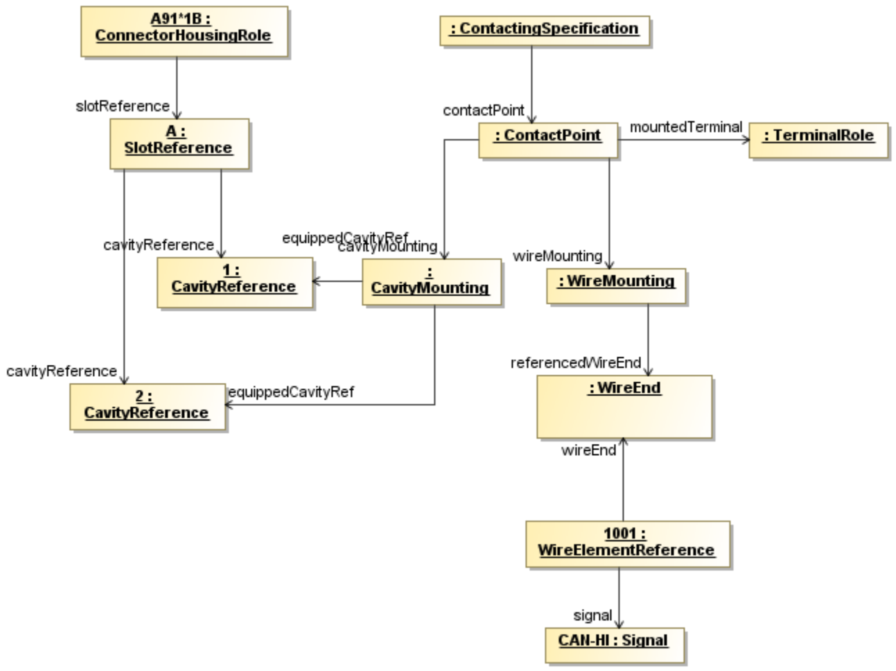


Figure 45: Bridge Terminal

A bridge terminal is a terminal that has a wire crimped on it and which occupies more than one cavity (short-circuited). On the side of the wire it is the same as a standard contact. On the side of the cavities it simply references more than one cavity.

5.1.5 Coax Contact

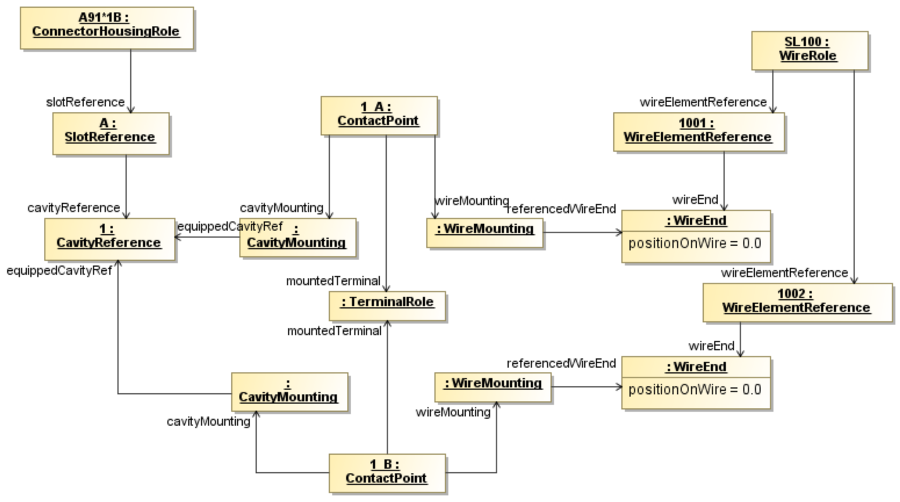


Figure 46: Coax Contact

The diagram displays the proper definition of a coax contacting. In the case of coax contact one single terminal is used, but two different electrical potentials are connected to it. Therefore two ConcactPoints are required, because one ContactPoint can only be used for one electrical potential (see the definition of a ConcactPoint).

Both ContactPoints reference the same occurrence of the terminal (TerminalRole) and use the Cavity. Each ConcatPoint mounts a single WireElement to the TerminalRole. In this example the two WireElements belong to the same multi-core wire.

In order to make the example more clearly, the next figure displays the definition of such a “coax-cavity” in an EEComponent.

5.1.6 Coax Cavity

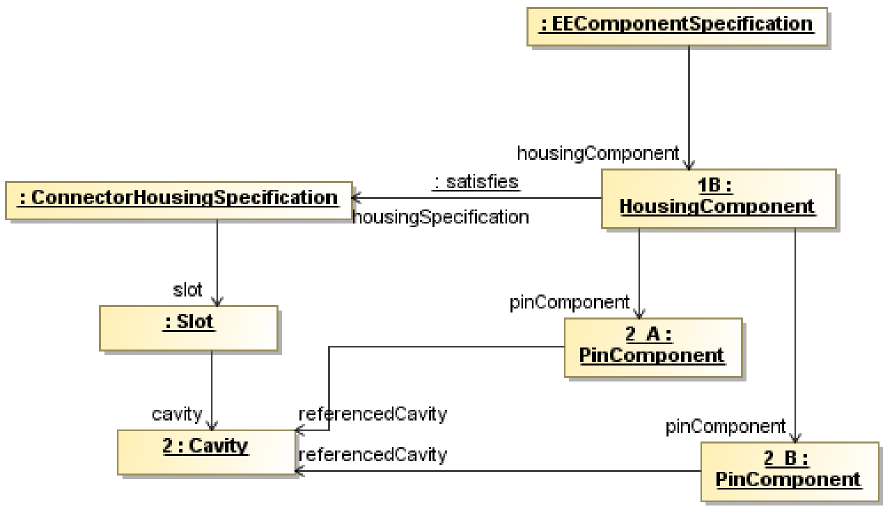


Figure 47: Coax Cavity

The HousingComponent of an EEComponent defines on one hand the pins (electrological relevant information) in this HousingComponent and on the other hand a ConnecorHousingSpecification (the layout and design of the HousingComponent). The PinComponents are then positioned in the cavities. In the case of a coax contact, two PinComponents (the different electrical potentials) are placed in one cavity.

5.2 *Mating Specification*

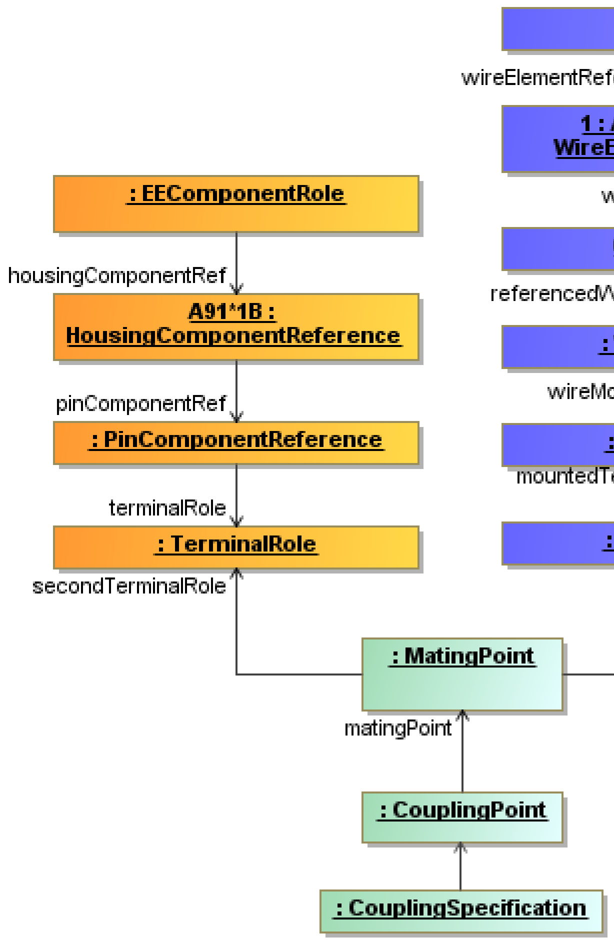


Figure 48: Mating Specification

The contacting defines in the VEC the relation of a terminal, a wire, a seal and a cavity in a connector on the harness side. The coupling / mating can be used to connect the harness side to another harness (in the case of an inliner) or to an ECU. The figure above shows a simplified example of the connection between an ECU (highlighted in orange) and a wiring connection (highlighted in blue). The MatingPoint simply connects the two TerminalRoles on each side. This method is used for example in a wiring definition, where the concrete connector is not yet known.

For this reason this example omits the geometric aspects of a coupling (connector housing, slots, cavities).

6 Complex Part Descriptions

This section is about the definition of Harnesses, Modules and Assemblies.

6.1 Module

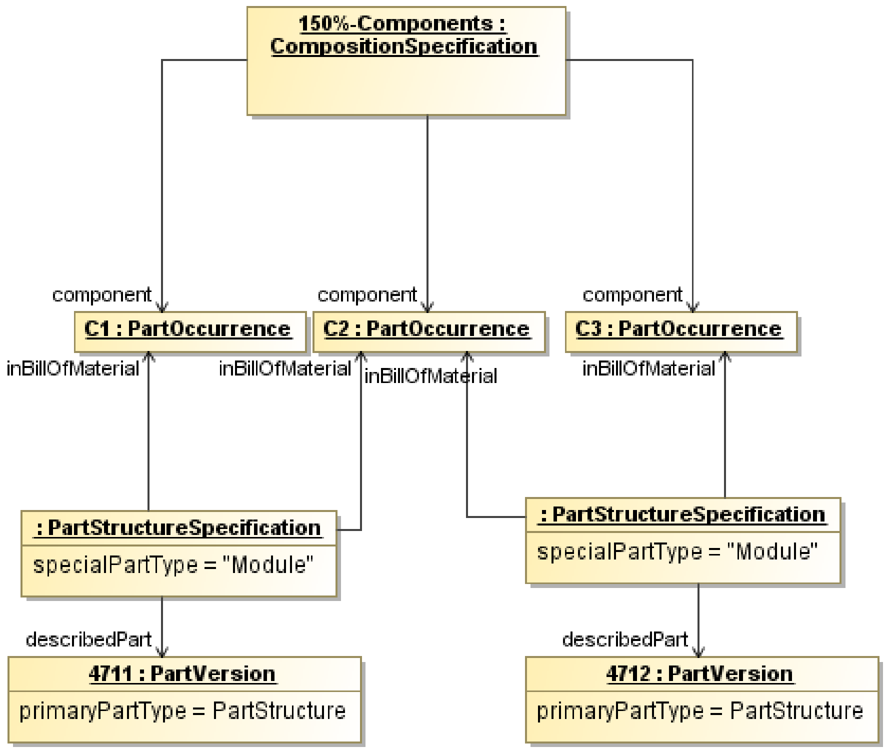

150%-Components : CompositionSpecification
component
component
component
C1 : PartOccurrence
C2 : PartOccurrence
C3 : PartOccurrence
inBillOfMaterial
inBillOfMaterial
inBillOfMaterial
inBillOfMaterial
: PartStructureSpecification
specialPartType = "Module"
: PartStructureSpecification
specialPartType = "Module"
describedPart
describedPart
4711 : PartVersion
primaryPartType = PartStructure
4712 : PartVersion
primaryPartType = PartStructure

Figure 49: Module

The figure shows how a module can be defined in a 150% environment. A module basically consists of a number of PartOccurrences. In a 150%-Harness the PartOccurrences can be shared between modules. Therefore a single container is needed, which defines a PartOccurrences that belong to the Harness, independent from the Module. This is done by a CompositionSpecification. The PDM information of a module is stored in the PartVersion. The PartOccurrences belonging to the Module (the “Bill Of Material”) are defined by a PartStructureSpecification.

6.2 Harness

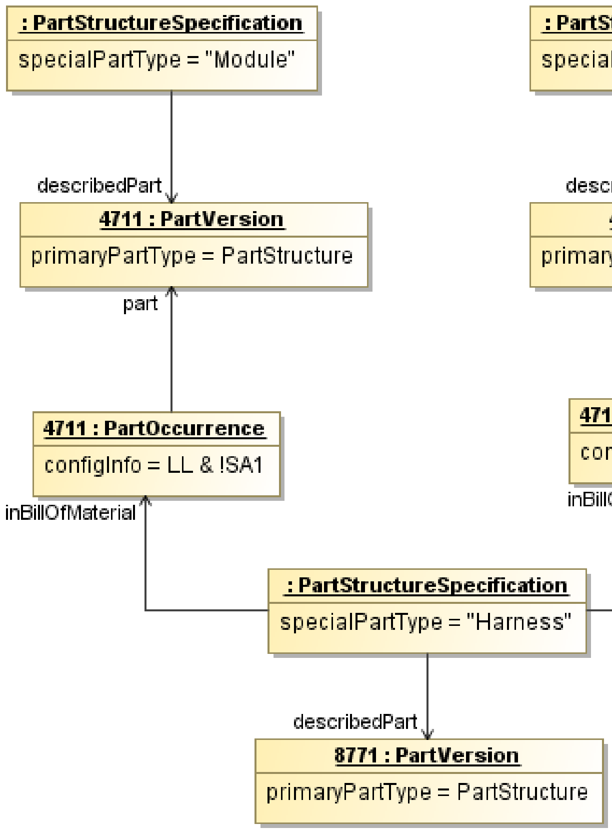

: PartStructureSpecification
specialPartType = "Module"
: PartStructureSpecification
specialPartType = "Module"
describedPart
describedPart
4711 : PartVersion
primaryPartType = PartStructure
4712 : PartVersion
primaryPartType = PartStructure
part
part
4711 : PartOccurrence
configInfo = LL & !SA1
4712 : PartOccurrence
configInfo = LL & SA1
inBillOfMaterial
inBillOfMaterial
: PartStructureSpecification
specialPartType = "Harness"
describedPart
8771 : PartVersion
primaryPartType = PartStructure

Figure 50: Harness

In the philosophy of the VEC a harness is quite the same as a module. A module is formed by a set of occurrences of components. A harness is formed by a set of occurrences of modules. This approach has the advantage that the same module can be used in different harnesses with for example different VariantConfigurations. The figure shows a simple configuration of a 150%-harness. The harness has a PartStructureSpecification, which defines the BOM of the Harness. This BOM contains the two PartOccurrences of the modules from the previous example. Each PartOccurrence can carry a VariantConfiguration. Since the two modules share a component, they are mutually exclusive (see the different VariantConfiguations).

6.3 Assemblies

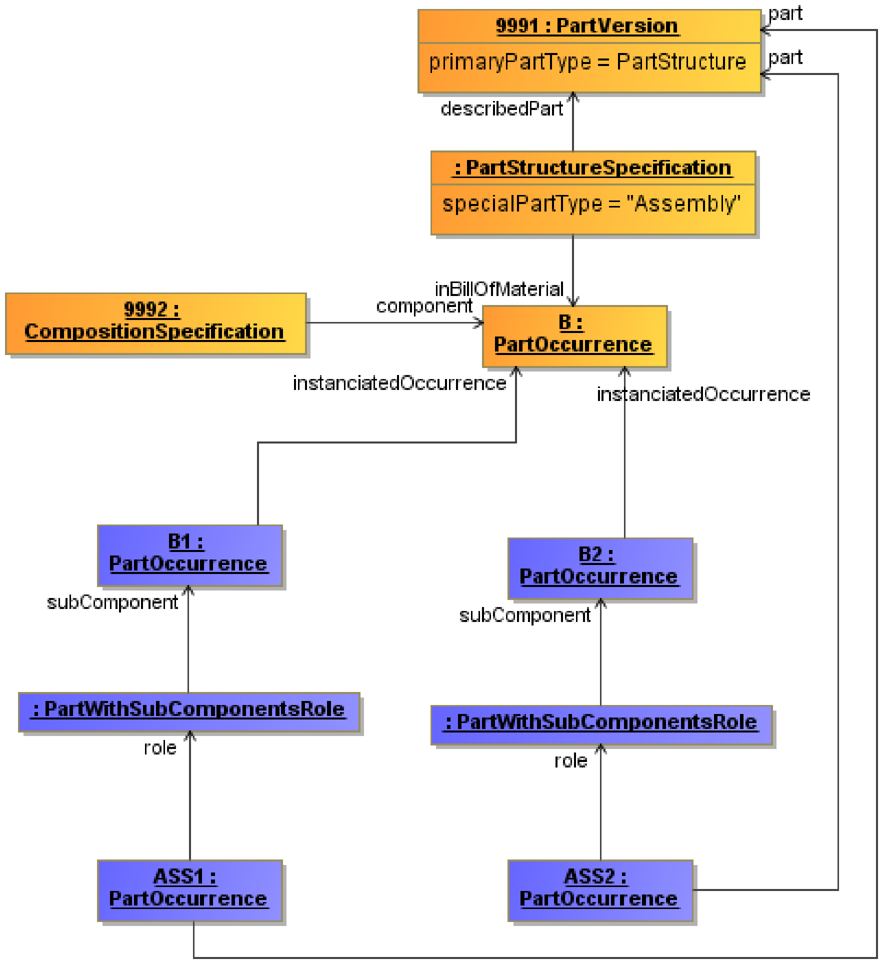


Figure 51: Assemblies

The figure above displays the usage of assemblies in the VEC. Assemblies are more complex components, which are built out of a number of other components (e.g. a prebuild wire with connectors attached to it). There are multiple reasons, why the inner structure of an assembly is relevant, but the most important one is that the inner components are needed to describe the correct usage of the assembly in the harness. For example in the case of the prebuild wire, the connectors are needed to define their position in the topology and the wire is needed to define the routing of the assembly through the topology.

In the upper area of the figure (highlighted in yellow) the master data of the assembly is shown (for simplification it contains only one PartOccurrence “B”). When the assembly is used in a Harness it is instantiated (highlighted in blue). In the example the assembly is instantiated twice (PartOccurrence ASS1 & ASS2), since an assembly can be used multiple times in the same context. All subcomponents of the the assembly **must** be instantiated as well (PartOccurrence B1 & B2). These PartOccurrences carry a reference to the PartOccurrence in the Part Master Data they represent. The instantiated PartOccurrences are used to define the concrete usage of the Assembly. It is also possible, that the concrete usage has properties different to the represented occurrence. The most common use case for this is the naming of the PartOccurrence. Another use case, which is not so obvious, is for example that a prebuild wire can have only one connector attached to it and the other side is left open. When the wire is used, it is cut to needed length and then contacted on the open side. In this case the wire length of the instantiated wire would be different to master data definition of the assembly.

7 External Mapping

7.1 Simple External Mapping

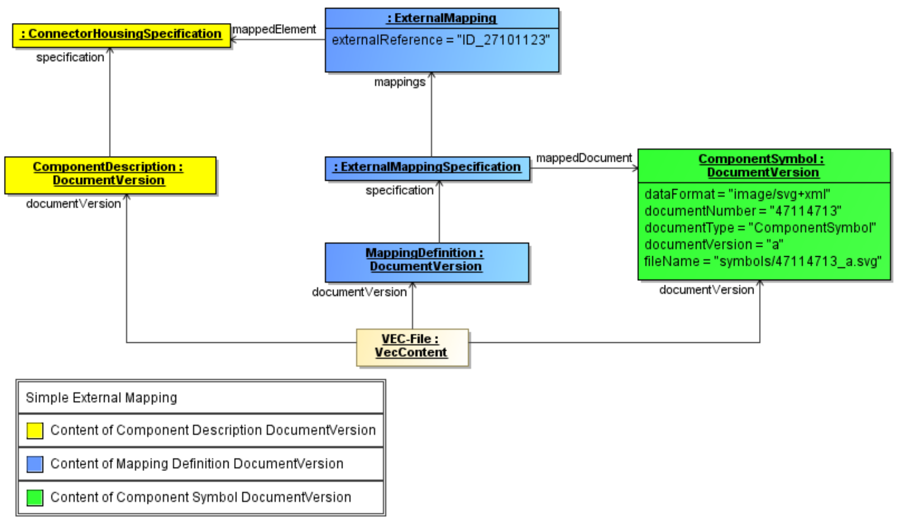

: ConnectorHousingSpecification
mappedElement
: ExternalMapping
externalReference = "ID_27101123"
specification
mappings
ComponentDescription : DocumentVersion
: ExternalMappingSpecification
mappedDocument
ComponentSymbol : DocumentVersion
dataFormat = "image/svg+xml"
documentNumber = "47114713"
documentType = "ComponentSymbol"
documentVersion = "a"
fileName = "symbols/47114713_a.svg"
documentVersion
specification
MappingDefinition : DocumentVersion
documentVersion
documentVersion
VEC-File : VecContent
Simple External Mapping
Content of Component Description DocumentVersion
Content of Mapping Definition DocumentVersion
Content of Component Symbol DocumentVersion

Figure 52: Simple External Mapping

The Diagram shows a simple example for the usage of the external mapping mechanism. The elements highlighted in yellow represent the actual information described by this VEC instance. The elements highlighted in blue are defining the concrete mapping and the *DocumentVersion* highlighted in green represents the link to the mapped document (in this case a SVG-symbol).

The *ExternalMapping* element in this example defines that a representation of the referenced *ConnectorHousingSpecification* can be found in the SVG-symbol under the key “ID_27101123”.

The actual content data of the VEC-file (highlighted in yellow) and the mapping information is separated into two different *DocumentVersion* elements. This means even though both information are contained in the same VEC-file, from the perspective of a versioning mechanism they are clearly separated.

8 Systems Schematic

The following section contains examples for the usage of different parts of the VEC required for the mapping of a specific use case, the “Systems Schematic”.

8.1 Variant Management for ECUs

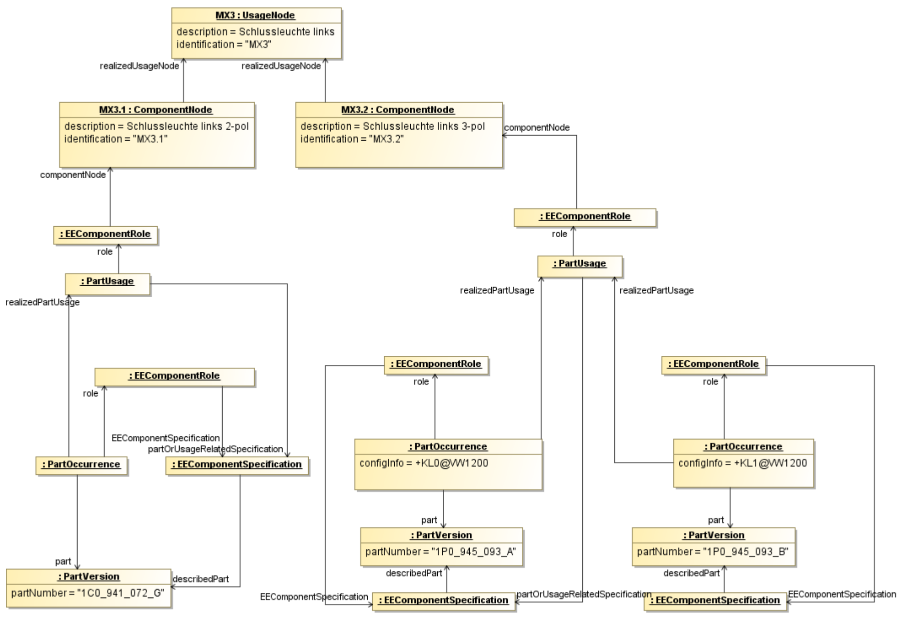


Figure 53: Variant Management for ECUs

This example demonstrates how the variant management can be handled in the systems schematic on different levels of abstraction.

The top most element is the usage node. It defines an abstract position / function in the vehicle. In the example it is the back light on the left hand side (named “MX3”). This function can be realized by two different electrological variants (interfaces). These variants are represented by ComponentNodes. In the example there is one variant with two pins (MX3.1) and one variant with three pins (MX3.2). On a more concrete level these interfaces can be satisfied by one or more EE-components (alternatives). These EE-Components are defined by PartVersion with a EEComponentSpecification. In order to define restrictions a corresponding PartOccurrence with a VariantConfiguration can be defined.

The PartUsages in the example are needed for to reasons:

1. They serve as a container to group the different possible alternatives (“realizedPartUsage”).
2. It is necessary to declare one of the EEComponents as the representative of all alternatives of a variant. This is done by the reference between the PartUsage and the corresponding EEComponentSpecification.

9 Packaging

9.1 Indexing of VEC Package

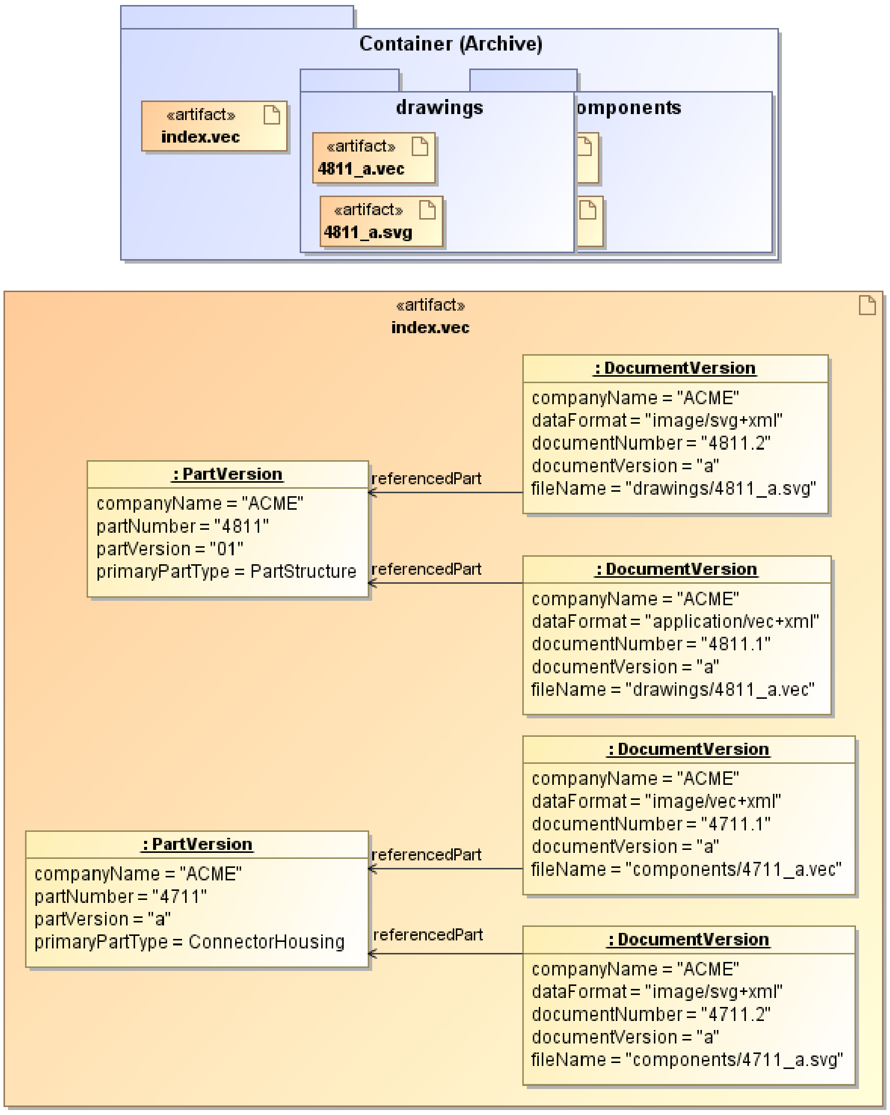


Figure 54: Indexing of VEC Package

This example explains how more than one VEC file can be packaged together with external documents, for example symbols of the specified components. In this case the VEC shall be packed within an archive together with the external documents (according to the current published VEC recommendation) and an index file has to be provided.

The upper half of the diagram illustrates the file structure within the archive. In the root there is a 'index.vec' file that describes the content of the archive. The simplified content of this index file is illustrated in the lower half of the diagram.

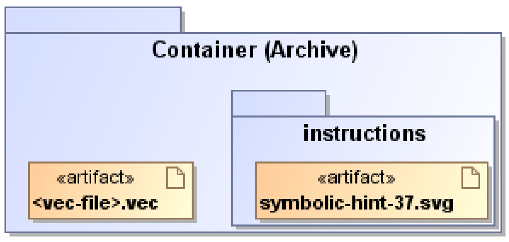
For the example, the archive consists of following files:

- index.vec: Describes the content of the archive.
- A harness with the part number 4811 specified by a VEC-File (4811_a.vec) and a 2D SVG representation of the harness (4811_a.svg).

- A component description of connector housing (part number: 4711) specified with a VEC-File (4711_a.vec) and component symbol (to be used in the 2D-drawing) defined with a SVG-Symbol (4711_a.svg).

In the VEC (especially in the index.vec) a DocumentVersion object is created for each external document (see lower half of the diagram). This DocumentVersion object references the PartVersion to which it is related. Beside all meta information about the document, the attribute dataFormat of the DocumentVersion specifies the format of the external document as a MIME-Type. The attribute fileName is a path relative to VEC-File pointing to the external document in the archive

9.2 External Installation Instructions



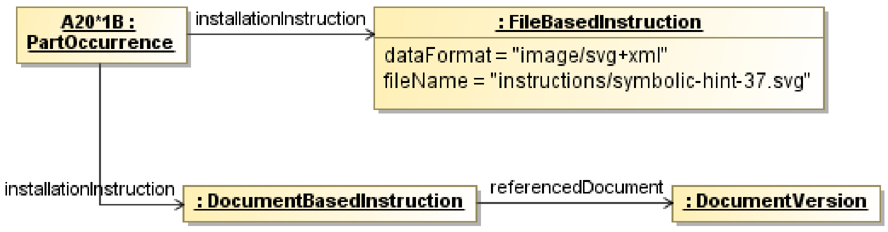


Figure 55: External Installation Instructions

The usage of file-based installation instructions is quite similar to the described approach for external documents. The FileBasedInstruction defines a pointer to the file packaged together with the VEC-file in the container and is referenced by the PartOccurrence.

The same effect could be achieved if a DocumentBasedInstruction is used, pointing to an external document (defined as described in the section before).

Important: The difference between the two approaches is that for the DocumentBasedInstruction a DocumentVersion is required. This means that the external file must be an official document with a document number, an appropriate versioning and so on. The FileBasedInstruction can point to any file needed. **It is not a valid approach to use the FileBasedInstruction, if the external file is a valid document.**