Document Title	Specification of Interoperability of AUTOSAR Tools
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	204
Document Classification	Auxiliary

Document Version	2.2.1
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History

Date	Version	Changed by	Description
2013-03-31	2.2.1	AUTOSAR Release Management	• Editorial changes
2013-01-18	2.2.0	AUTOSAR Administration	• Add formal specification items • Support for the processor manifest • Support for roles and rights
2011-10-31	2.1.0	AUTOSAR Administration	• Editorial changes including tagged specification items • Improved recommendation of usecases for AUTOSAR files • Refined definition of XML serialization

2009-11-30	2.0.0	AUTOSAR Administration	• Refined scope to AUTOSAR tools • Added R4 aspects (Variant handling, splittable, relative reference) • More details on XML serialization • More details on error reporting • More details on merging • Removed requirements on AUTOSAR-Products (Meta-model and Schema)
2008-06-23	1.2.1	AUTOSAR Administration	• Legal disclaimer revised
2007-11-14	1.2.0	AUTOSAR Administration	• Added description on how to merge models • Removed dependencies to no longer existing documents • Added requirements on extension mechanism • Added requirements on handling / exchanging errors • Document meta information extended • Small layout adaptations made
2007-01-31	1.1.0	AUTOSAR Administration	• NonSplitableElements are the minimum granularity of most AUTOSAR descriptions. In case a smaller granularity is required, the meta-model may explicitly define SplitableElements • Legal disclaimer revised • Release Notes added • "Advice for users" revised • "Revision Information" added
2005-12-12	1.0.0	AUTOSAR Administration	• Initial release

Table of Contents

Known Limitations

- Requirements were not updated
- Chapter 6 (Compliance) is incomplete. It specifies aspects but not specific compliance criteria.

References

[1] Requirements on Interoperability of AUTOSAR Tools
AUTOSAR_RS_InteroperabilityOfAutosarTools

[2] Methodology
AUTOSAR_TR_Methodology

[3] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate

[4] Model Persistence Rules for XML
AUTOSAR_TR_XMLPersistenceRules

[5] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[6] Specification of ECU Resource Template
AUTOSAR_TPS_ECUResourceTemplate

[7] System Template
AUTOSAR_TPS_SystemTemplate

[8] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate

[9] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration

[10] Unified Modeling Language: Superstructure, Version 2.0, OMG Available Specification, ptc/05-07-04
http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf

[11] Unified Modeling Language OCL, Version 2.0, OMG Available Specification, ptc/05-06-06
http://www.omg.org/cgi-bin/apps/doc?ptc/05-06-06.pdf

[12] Technological Spaces: an Initial Appraisal
http://www.sciences.univ-nantes.fr/lina/atl/www/papers/PositionPaperKurtev.pdf

[13] Extensible Markup Language (XML)
http://www.w3.org/TR/xml11

[14] XML Schema 1.1
http://www.w3.org/XML/Schema

[15] Virtual Functional Bus
AUTOSAR_EXP_VFB

[16] Meta Model
AUTOSAR_MMOD_MetaModel

[17] Container Catalog XML Model Specification

http://www.asam.net

[18] OASIS Web Site: TR 9401:1995 - Entity Management
http://www.oasis-open.org/specs/a401.htm

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

1 Introduction

This document describes various aspects of interoperability of AUTOTSAR tools in order to

- support proper implementation of AUTOSAR data exchange
- contribute to (but not fulfill) requirements on AUTOSAR Tools
- specify more details about XML handling
- discuss essential aspects for tool interoperability
- provide more information about how to utilize particular Meta-Model concepts

The dependency relationships of this document are depicted in Figure 1.1.

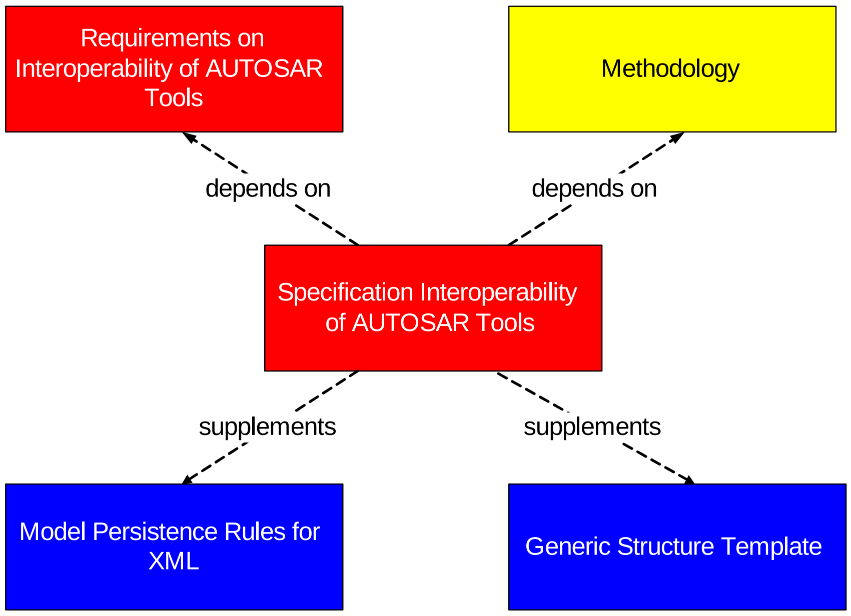
Requirements on Interoperability of AUTOSAR Tools
Methodology
depends on
depends on
Specification Interoperability of AUTOSAR Tools
supplements
supplements
Model Persistence Rules for XML
Generic Structure Template

Figure 1.1: DocumentDependencies

According to Figure 1.1, this document depends on the "Requirements on Interoperability of Authoring Tools" [1] and the "Methodology" [2].

This document defines requirements on AUTOSAR tools. Thereby it supplements the document "Generic Structure Template" [3] and the document "Model Persistence Rules for XML" [4] under following aspects:

- "Model Persistence Rules for XML" describes the M2 aspects while "Interoparability of AUTOSAR tools" handles M1 (AUTOSAR XML description) aspects.
- "Generic Structure Template" describes meta-model facilities while "Interoperability of AUTOSAR tools" handles implementation specific aspects.

1.1 Classification of AUTOSAR Tools (non-normative)

The AUTOSAR methodology model [2] describes the major steps of a development of a system with AUTOSAR: From the system level to the generation of an ECU executable. It describes the dependencies of work products and tasks.

An AUTOSAR tool can support one or more tasks of the AUTOSAR methodology. In other words, the term *AUTOSAR Tool* refers to all tools that support the tasks of creation, modification and interpretation of AUTOSAR models which describe a system and its configuration as defined in e.g. the

- Generic Structure Template [3]
- Software-Component Template [5],
- ECU Resource Template [6],
- System Template [7],
- Basic Software Module Description Template [8],
- Specification of ECU Configuration [9],

Thus all AUTOSAR tools deal in some way with AUTOSAR XML descriptions (i.e. the XML representation of AUTOSAR models, see [4]). Depending on the nature of the interaction with XML, three kind of AUTOSAR tools can be distinguished, as depicted in Figure 1.2:

- *AUTOSAR Importer Tools* used to create AUTOSAR models (as XML descriptions) by importing **non-**AUTOSAR artifacts. Note that an importer tool may also be integrated in an AUTOSAR Authoring Tool.
- *AUTOSAR Authoring Tools* used to create and modify AUTOSAR models (as XML descriptions)
- *AUTOSAR Converter Tools* used to produce new AUTOSAR XML descriptions by converting information from existing ones
- *AUTOSAR Processor Tools* used to produce **non-**AUTOSAR artifacts by processing AUTOSAR XML descriptions

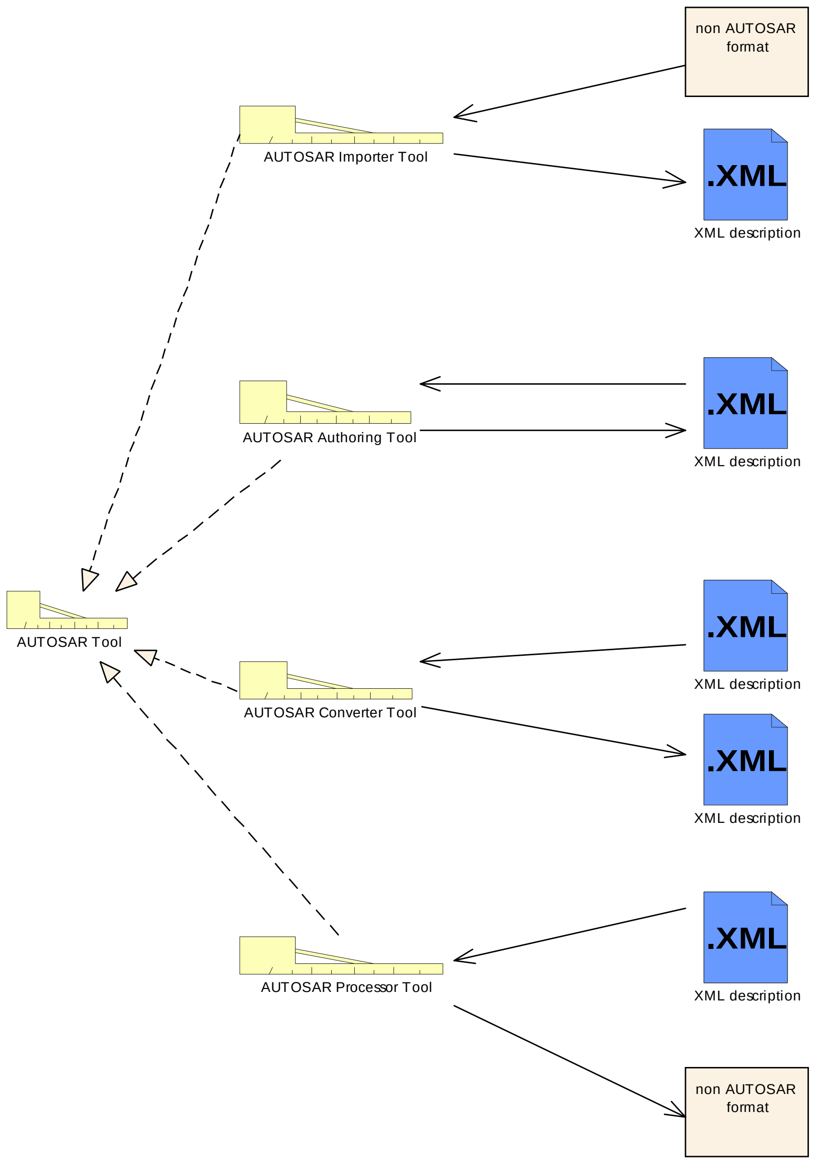

Figure 1.2: Categories of AUTOSAR Tools

A tool can also act as a combination of these three types, e.g. the RTE generator can produce code and XML descriptions (e.g. of its own implementation aspects) in one step, acting as a combination of a converter and a processor tool.

Since from these three types only the authoring can modify an already existing AUTOSAR model, in general most requirements for interoperability apply for authoring tools.

Figure 1.3 sketches some descriptions that can be maintained by AUTOSAR authoring tools within the AUTOSAR methodology (for a detailed description of the notation see [2]). The figure shows only some examples out of all the task which can be performed on these artifacts.

The formal description of AUTOSAR software-components does not include a complete formal description of the behavior of the software-component. The latter is intentionally left to dedicate Behavior Modeling Tools (BMT).

It is therefore necessary to bridge the gap between a software-component model and the corresponding behavior model created by a particular BMT. This task is carried out by the "Coupling Tool" mentioned in Figure 1.3.

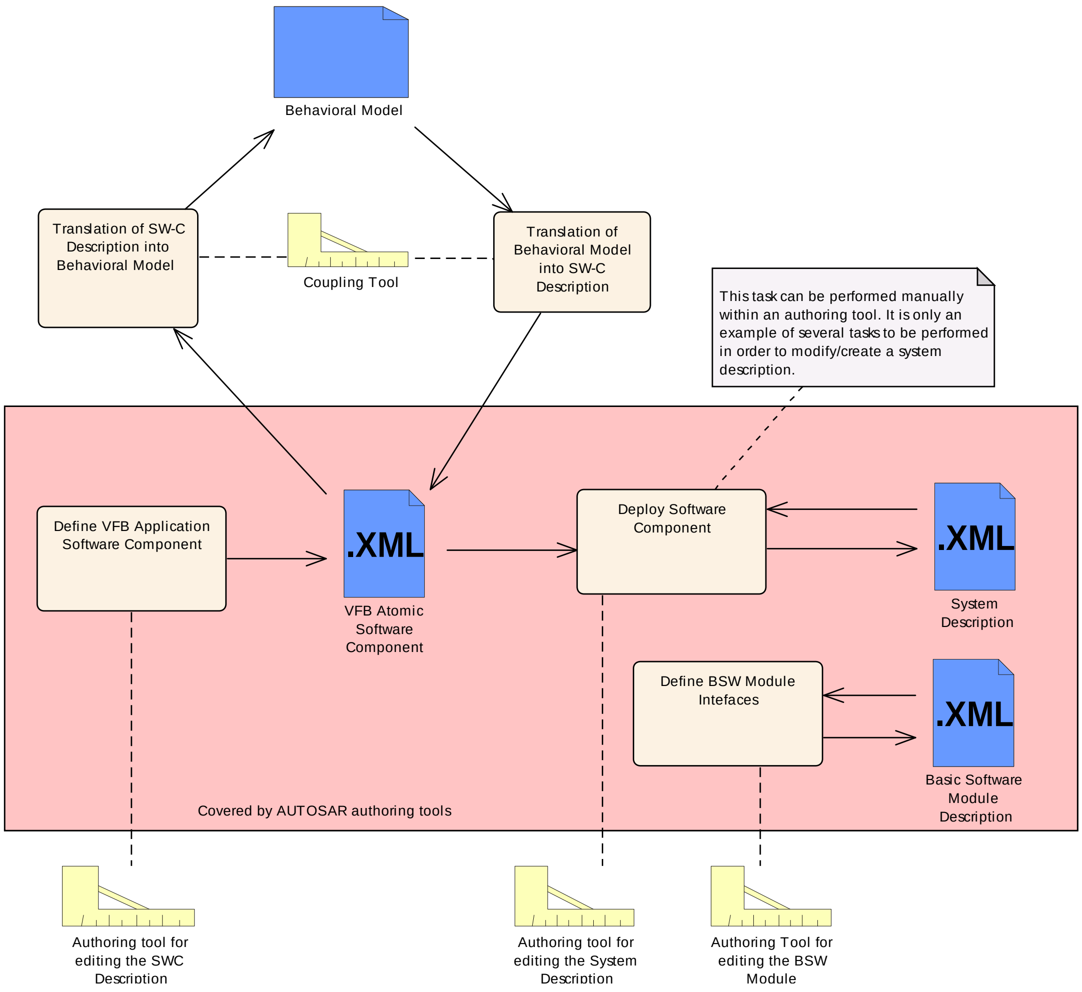
Behavioral Model
Translation of SW-C Description into Behavioral Model
Coupling Tool
Translation of Behavioral Model into SW-C Description
This task can be performed manually within an authoring tool. It is only an example of several tasks to be performed in order to modify/create a system description.
Define VFB Application Software Component
.XML
VFB Atomic Software Component
Deploy Software Component
.XML
System Description
Define BSW Module Intefaces
.XML
Basic Software Module Description
Covered by AUTOSAR authoring tools
Authoring tool for editing the SWC Description
Authoring tool for editing the System Description
Authoring Tool for editing the BSW Module

Figure 1.3: Examples for tasks of authoring tools including coupling between a Behavioral Model and AUTOSAR Models

Figure 1.4 shows some examples of “downstream” XML artifacts and tasks, i.e. those used to derive an ECU configuration out of a system description (this figure only gives a rough overview, for the full picture refer to [2]).

In order to create the downstream artifacts, converter tools come into play, also a combination of converter and authoring tools.

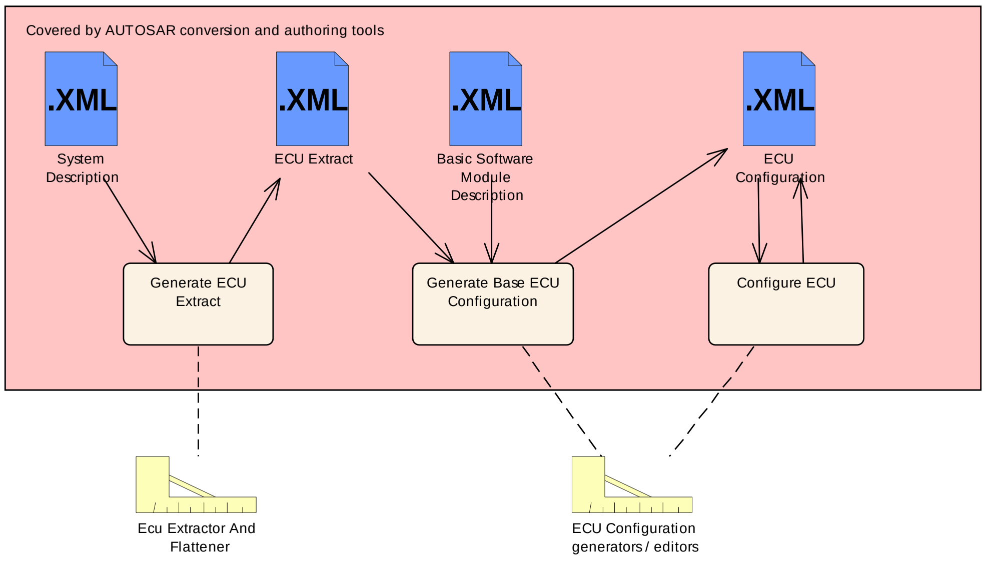
Covered by AUTOSAR conversion and authoring tools
.XML
System Description
.XML
ECU Extract
.XML
Basic Software Module Description
.XML
ECU Configuration
Generate ECU Extract
Generate Base ECU Configuration
Configure ECU
Ecu Extractor And Flattener
ECU Configuration generators / editors

Figure 1.4: Examples for tasks of converter and authoring tools handling “downstream” artifacts

1.2 Origins and goals (non-normative)

Whenever data is exchanged between different parties they need to agree on a common understanding about the wording and the semantics. Otherwise there will be misunderstandings. This observation applies to communication between different tools as well.

AUTOSAR formally defines the structure and semantics of data by means of UML[1] class diagrams, semantic constraints in the template specifications as well as by OCL[2].

In addition to a common data exchange language (specified by the AUTOSAR XML schema) further issues need to be considered in order to support a successful communication.

This document points out potential problems when exchanging models between different tools and companies and defines strategies on how these problems could be solved.

Based on [1] and a set of use-cases (chapter 2) requirements on AUTOSAR tools are defined.

All requirements described in this document are summarized in chapter 6. These requirements relate to all AUTOSAR tools used to create or interpret AUTOSAR xml descriptions. Some requirements are specific to AUTOSAR authoring tools.

This document is structured as follows:

[1]UML: Unified Modeling Language [10]
[2]OCL: Object Constraint Language [11]

- Chapter 1 "Introduction" (this chapter) gives an overview of the documents that deal with AUTOSAR authoring tools.
- Chapter 1.4 "Requirements Tracing" lists the requirements on this document and associates the chapters where these requirements are addressed.
- Chapter 2 "Use-Cases (non-normative)" gives some examples on information interchange between AUTOSAR tools. The intention is to point out potential problems while using different tools.
- Chapter 3 "Use-case tracing (non-normative)" shows which use-cases are covered in which sections of this document.
- Chapter 4 "Basic concepts" describes some basic concepts which are used in the following chapters.
  - Subchapter 4.1 "Data representation" explains the relationship between AUTOSAR template, AUTOSAR XML schema, AUTOSAR models, AUTOSAR XML descriptions, etc.
  - Subchapter 4.2 "Abstraction levels of information exchange" shall help to understand the tasks which need to be done when exchanging AUTOSAR models via XML. These tasks are illustrated by different levels. Note: These level are introduced as a concept. This does does not imply that each AUTOSAR tool must exactly implement this as its architecture.
- Chapter 5 "Requirements on AUTOSAR tools" explains aspects for tool interoperability and defines requirements on AUTOSAR tools. E.g.:
  - Integration of tools which do not support the full set of information defined in the meta-model: Those tools only need to import and export the information that can be internally represented.

    Merging the results of the input information with the output exported by the authoring tool results in an updated overall model. All authoring tools are required to support this merge for the set of information that is internally supported.
  - Migration between different versions of the AUTOSAR meta-model: Each tool should support manual or automatic upgrades of AUTOSAR XML descriptions which were created with respect to the last major version of the AUTOSAR meta-model.
  - Support for concurrent modeling of AUTOSAR systems: e.g. each authoring tool should support working on incomplete AUTOSAR models. It should be possible to merge several AUTOSAR models together.
- Chapter 6 "Compliance" discusses the compliance of AUTOSAR tools and summarizes the requirements defined in this document: A tool may be called AUTOSAR compliant if it implements all mandatory requirements on AUTOSAR tools defined in this document.

Compliant tools should be able to be used within the AUTOSAR methodology. However, a seamless exchange of AUTOSAR XML descriptions between different tools is only possible if they support the same set of information.

1.3 Document Conventions

Meta-model terms are typeset in mono spaced font, e.g. `PortPrototype`. As a general rule, plural forms of these 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 meta-model respectively the XML Schema.

Each requirement on AUTOSAR tools is defined as a table. The structure of the tables is as follows:

[traceid] Headline of Requirement ⌈

Description:	Detailed description
Rationale:	Why is this requirement important, what its omission could cause?
Use Case:	A scenario that makes the requirement necessary or useful
Dependencies:	References to other requirements which this requirement depends on
Supporting Material:	References to other documents, models etc.

⌋

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. Note that the requirement level of the document in which they are used modifies the force of these words.

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

place requires it or because the vendor feels that it enhances the product while another vendor may omit the same item.

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

1.4 Requirements Tracing

The following table references the requirements specified in [1] (**[RS_IOAT_00001]**, **[RS_IOAT_00002]**) respectively the use cases specified in this document and links to the fulfillments of these.

Requirement	**Description**	**Satisfied by**
[RS_IOAT_00001]	Support data exchange	[TR_IOAT_00071] [TR_IOAT_00072]
[RS_IOAT_00002]	Standardize the handling of errors in AUTOSAR models	[TR_IOAT_00065]
[UC_IOAT_00001]	Integrate extracts from an AUTOSAR model of an OEM passed for further refinement and implementation to a supplier	[TR_IOAT_00035] [TR_IOAT_00063] [TR_IOAT_00064]
[UC_IOAT_00002]	Dealing with changes of the AUTOSAR meta model over time	[TR_IOAT_00005] [TR_IOAT_00066]
[UC_IOAT_00004]	Allowing for concurrent work on the same mode	[TR_IOAT_00062] [TR_IOAT_00067] [TR_IOAT_00074]
[UC_IOAT_00005]	Usage within the different steps of top-down functional development	[TR_IOAT_00035] [TR_IOAT_00065]
[UC_IOAT_00006]	Support for direct exchange of AUTOSAR models in a tool-chain	[TR_IOAT_00065]
[UC_IOAT_00008]	An AUTOSAR model and related artifacts are shipped from one party to another.	[TR_IOAT_00036] [TR_IOAT_00062] [TR_IOAT_00065] [TR_IOAT_00073]
[UC_IOAT_00010]	Handling of identical double definitions	[TR_IOAT_00063] [TR_IOAT_00064]

2 Use-Cases (non normative)

This chapter describes use-cases for the interoperability of AUTOSAR tools. The intention of these use cases is to point out potential problems that might occur when exchanging AUTOSAR models (represented as AUTOSAR XML descriptions) in a development process.

It is NOT intended to define a standardized AUTOSAR process. **The use-cases are EXAMPLES that are intended to highlight potential problems while exchanging AUTOSAR models**.

Each use-case defined in this document has its unique identifier starting with the prefix "ATUC" (meaning AUTOSAR Tool Use Case).

Please note the different levels between the use cases described here and the use cases (also called "capability patterns") described in the AUTOSAR methodology model (see [2]).

The use cases in the methodology focus on the logical tasks and their work products and do not address the aspects of using different tools, as the use cases do which are listed below.

The same task of the methodology may be performed by various AUTOSAR tools. On the other hand one particular AUTOSAR tool may be used to perform several different tasks. When different tools are used in a project, the logical data flow described by the methodology must nonetheless be provided.

2.1 Usage within the different steps of top-down functional development

[UC_IOAT_00005] Usage within the different steps of top-down functional development ⌈ When developing a system using the top-down approach, the AUTOSAR models are first initiated as outlines, then refined, and updated by the OEM or the supplier through successive iteration loops as the development of the network and the ECUs progresses. ⌋

This development process includes for example the following steps which are related to tool interoperability:

- The initial AUTOSAR model may be automatically generated out of an existing proprietary database or created manually from scratch.
  (Action: conversion of data from a proprietary database to the AUTOSAR format)
- The incomplete result may be edited by another tool and/or person.
  (Action: exchange of incomplete models between tools within a company)
- The AUTOSAR model may be created to a given level of granularity by the OEM and then passed over to another department or to a supplier.

(Action: exchange of partial models between tools that are used in different companies)

- It may be the case that only a subset of the whole AUTOSAR model is passed to a supplier. The supplier may need to make sure that all required information is available. The possible partitioning of an AUTOSAR model is defined using ≪atpSplitable≫. Therefore AUTOSAR tools shall be able to handle partial models (e.g. with dangling references)
(Action: extraction of an AUTOSAR model out of the full AUTOSAR model, so that the extracted model only contains the information that is required by another party; check if model was changed while it was sent to another party).

  This use-case has different aspects: partial model can mean:

  1. a subset of model elements (e.g. only some components)
  2. a subset of the specification of a particular model element (all components, but not all data of the components e.g. not Internal Behavior)
  3. a combination of these two: Only some components with only a subset of the component description.

- A supplier may be contracted to implement an AUTOSAR software component. The supplier needs to return a complete AUTOSAR model for the implemented component. The OEM might need to evaluate if the AUTOSAR model is complete in order to facilitate further processing in the AUTOSAR tool chain

  (Action: check if a model that is returned from another party only contains valid changes. E.g. only the AUTOSAR software component was changed. The interface descriptions were not changed).

- At some point in time some AUTOSAR models may need to be integrated / merged. Potential collisions need to be resolved. Two cases need to be distinguished:

  – “integrated” means that a component is incorporated into an existing system. Even if a component can be considered as a partial model of an system wide AUTOSAR model, there are still some tasks to be performed as defined in the AUTOSAR Methodology (e.g. define mappings). ([2])

  – “merged” means that a partial model is integrated into an existing model. The main motivation for this is concurrent modeling, variant handling, different responsibilities along e.g. a component development process.

  Please refer also to [UC_IOAT_00009], [UC_IOAT_00004], [UC_IOAT_00010]

2.2 Support for subcontracting

[UC_IOAT_00001] Integrate extracts from an AUTOSAR model of an OEM passed for further refinement and implementation to a supplier ⌈ Automotive systems are

developed by several companies. An OEM could develop a system until a given granularity is reached and then pass the further development to one or more suppliers.
⌋

For example, an OEM defines a coarse-grained architecture of software components, their interfaces, and connectors between them. This architecture is refined and implemented by some suppliers. The suppliers are not allowed to change any interfaces which were defined by the OEM.

Otherwise this would lead to problems during the integration phase. The OEM needs to find out which changes have been made on the models by the suppliers. Therefore, a tool that checks differences between models is required.

Additionally, a formal mechanism for explicitly describing which parts of a model may be modified or extended by suppliers could avoid misunderstandings and conflicts during integration of the results.

Authoring tools could evaluate the access rights and warn the user if he tries to modify elements he is not allowed to edit (the details are explained in [3]).

Such mechanisms can easily be based on proper distribution to sub-models (using the stereotype ≪atpSplitable≫. In this case the meta data in the ASAM catalog (see 5.3.1) can indicate the changed artifacts.

A more fine-grain control can be performed using specific Collections.

This use-case applies in particular to AUTOSAR authoring tools which are used to create and maintain AUTOSAR XML descriptions.

2.3 Support of different Versions of Meta-Model

[UC_IOAT_00002] Dealing with changes of the AUTOSAR meta model over time
⌈ The AUTOSAR meta-model and the derived AUTOSAR data exchange format will change over time. It SHALL be predictable whether tools (potentially with different underlying meta model versions) can exchange AUTOSAR models.

⌋

2.4 Concurrent modeling

[UC_IOAT_00004] Allowing for concurrent work on the same mode ⌈ A complete system can be represented as a big AUTOSAR model. Several co-workers, departments or even companies are concurrently working on parts of the model. The following sections describe some more detailed scenarios. ⌋

Concurrent development is restricted to `PackageableElement`s. It should always be clear who is responsible for a particular `PackageableElement`. The representation of this responsibility is not in the scope of AUTOSAR. It could be handled in the catalog.

In addition to this, concurrent development can be handled by split the work into artifacts such that one party can handle the artifact on his own. This is supported by application of the stereotype ≪`atpSplitable`≫. There is no more concurrency than provided by ≪`atpSplitable`≫.

2.4.1 Renaming model elements

Parties X and Y work on model A. Elements of the model are connected to elements of Model B, which is local to party X (see Figure 2.1).

- Party X has model A, which contains an element "BrakControl"
- Model A is passed on to party Y for further refinement (indicated by the ≪`trace`≫ arrow in the diagram)
- Party Y modifies the model and renames "BrakControl" to "BrakeControl"
- Party Y returns modified element to party X. Party X has to merge modified data. X faces a problem: Party X uses the element "BrakControl", which is no longer existent in the new model.

If party X identifies elements by their name, party X has no way to decide if "BrakControl" was deleted and a completely new independent element "BrakeControl" has been introduced or if "BrakControl" was renamed.

In the latter case, keeping all the original associations of "BrakControl" to other model elements would make sense, in the former it would not.

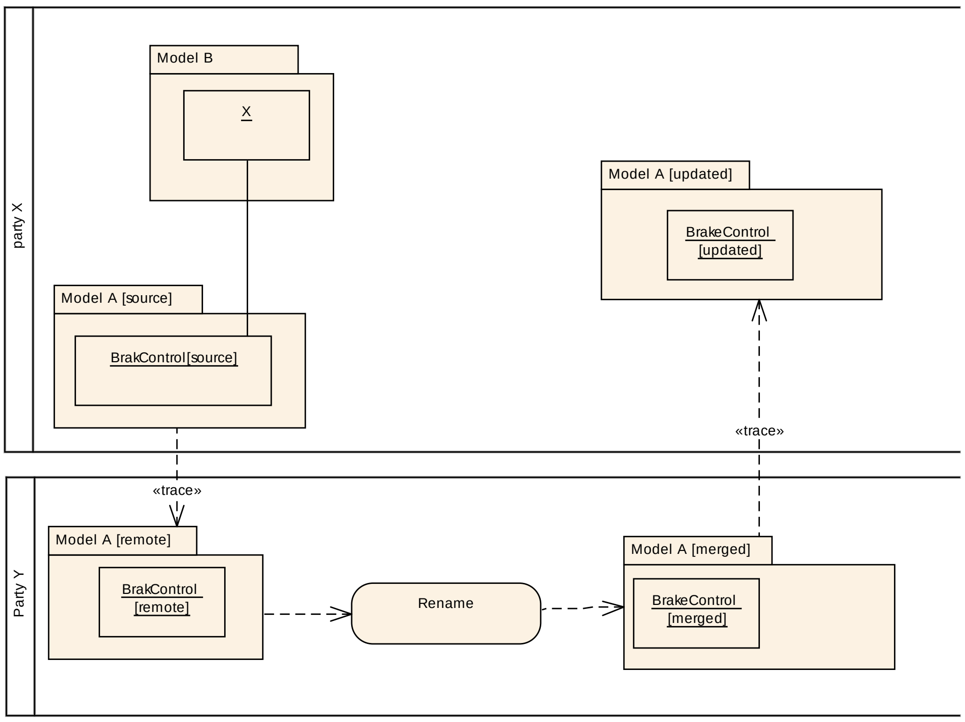

Figure 2.1: Concurrent modeling - renaming of elements

2.4.2 Updating of model elements

This scenario is similar to the renaming scenario; it differs only in the workflow (see Figure 2.2):

- Party X has the Model A, which contains an element "BrakControl"
- Model A is passed on to Party Y for further refinement (indicated by the "«trace»" arrow in the diagram
- Party Y uses the model and connects model elements to element of its own model B
- While party Y is using model A, party X detects a problem in its model, fixes it and wants to provide the updated model to party Y.
- Party Y has to merge the modified data. Y faces the same problems as in the first renaming scenario: Party Y uses the element "BrakControl", which is no longer existent in the new model

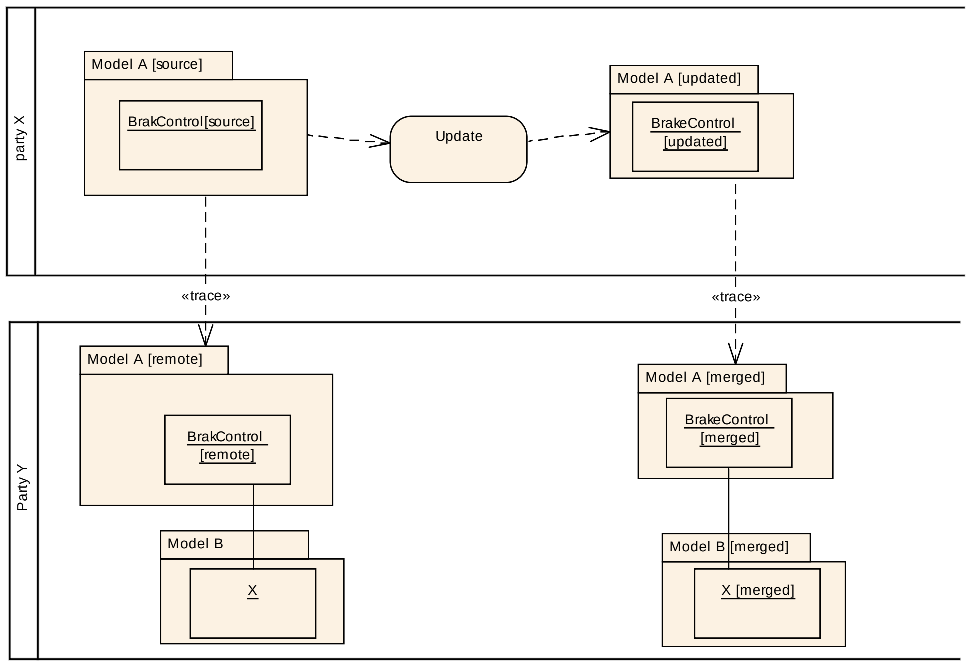

Figure 2.2: Concurrent modeling - updating of elements

2.4.3 Moving of elements from one namespace to another

If an element is moved from one AUTOSAR name space to another, this is basically the same as a rename, since model elements are identified by their fully qualified name which is the concatenation of all `shortName`s up to the root of the model.

This scenario is basically similar to the scenarios described in sections 2.4.1 and 2.4.2.

2.4.4 Parallel development of models

Several developers might create models in parallel. Each of them works on his local version of a model. At some point of time it turns out, that developer A needs some model elements that are in the responsibility of developer B.

It should be possible that developer A can create a reference to the elements of developer B, even if the content is not available in his local copy.

Another issue which might occur is that developer A and developer B both model the same content. The authoring tool should support merging the models of developer A and B. It should be able to detect potential conflicts.

It might also be the case that developer A and developer B are working on different variants in the same context, e.g. a low-end and a high-end capability which are targeted at different market segments. An authoring tool should allow the parallel development of variants, support merging these variants into a combined model and assist the user in creating appropriate variation points.

2.5 Direct exchange of AUTOSAR model in tool-chain

[UC_IOAT_00006] Support for direct exchange of AUTOSAR models in a tool-chain ⌈ This use case describes how information could be exchanged between authoring tools. In this use case each tool exports the AUTOSAR model to an XML description which is then directly imported by the next tool. ⌋

A scenario for a direct exchange is:

- An OEM might import some data from an existing database and create an initial AUTOSAR model using "Authoring tool 1".
- The result is extended by the "Authoring tool 2".
- An extract of the AUTOSAR model is passed to a supplier for further refinement.

This scenario implies that each tool in the tool chain is able to handle all information created by any other tool which was used in the chain before.

This exchange is not limited to file exchange but can also be performed using cut and paste. Regardless of the physical level, AUTOSAR XML description is the only standardized exchange format for model elements.

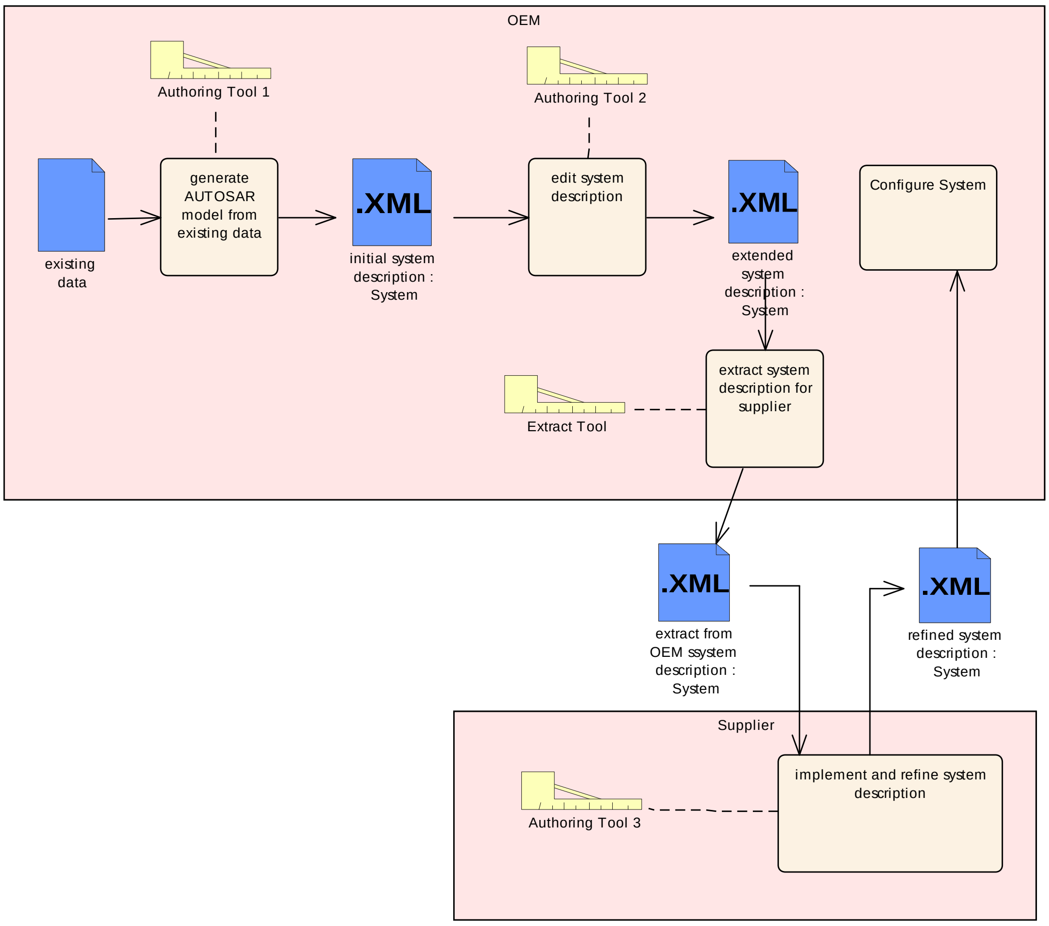
OEM
Authoring Tool 1
Authoring Tool 2
existing data
generate AUTOSAR model from existing data
.XML
initial system description : System
edit system description
.XML
extended system description : System
Configure System
extract system description for supplier
Extract Tool
.XML
extract from OEM ssystem description : System
.XML
refined system description : System
Supplier
implement and refine system description
Authoring Tool 3

Figure 2.3: Tool Chain

2.6 Shipment of AUTOSAR models and related artifacts

[UC_IOAT_00008] An AUTOSAR model and related artifacts are shipped from one party to another. ⌈ If two parties exchange an AUTOSAR model, the receiver needs to know the AUTOSAR model that can be shipped via several files has been received correctly.

As an example, OEM wants to send information to tier-1 supplier. OEM wants to lock certain model elements so that the tier-1 is not allowed to change them. The tier-1 needs to find out if all information has been transmitted correctly.

The meta-data for data exchange could additionally list further files that are not specified by AUTOSAR, e.g. specific model files of behavior modeling tools. ⌋

The following work flow describes how an AUTOSAR model could be handled, if additional meta information for data exchange is available:

- In addition to the AUTOSAR model itself, additional artifacts in electronic form need to be shipped as well, e.g. the component's object code or a model of a behavior modeling tool.
- The AUTOSAR model is very likely split into sub-models. The relevant artifacts and roles of the sub-model needs to be specified. This needs to be mutually agreed between the involved parties.
- In a collaborative exchange scenario, the sender also wants to submit version information and information about new / deleted artifacts.
- The sender might also want to add some meta-data that describes which parts of the AUTOSAR model may be changed and which are not allowed to be changed.

Workflow at the OEM's site In this case an OEM gathers a collection of model elements to be submitted to a supplier. The meta-data for data exchange is stored into a file when the collection is complete. This file could be called a manifest or catalog.

The OEM could create a specific folder in the model repository and names it after the characteristics of the information exchange. The catalog file is checked into this folder. Now, all versions of model files that are part of the exchange are shared into the folder.

As a result, the OEM gets a comprehensive description of the model interchange without touching the model files themselves. The catalog file could contain information about which parts of the AUTOSAR model are allowed to be changed.

Now the OEM repeats the same activity for model interchange with a different supplier who is responsible for refinement of another part of the AUTOSAR model and therefore the access rights for the second supplier are different.

Let's assume that the collection of model files submitted to the two suppliers is identical with respect to the version of the models. The OEM is now capable of recognizing that model files submitted to different suppliers are exactly identical although the access rights might be different.

Workflow at the supplier's site The supplier receives the catalog file and feeds it to the AUTOSAR authoring tool in use. The latter takes the catalog file as the basis for the actual import of AUTOSAR models. Access rights as well as other meta-information would most likely be taken over by the AUTOSAR authoring tool.

The supplier now implements the behavior of received `AtomicSwComponentType`s. The implementation of the behavior has an impact on the `Implementation` description of `AtomicSwComponentType`s. Therefore, the version of the `Implementation` must be changed. It is not subject of Interoperability to determine how and by whom the version is changed.

The supplier then exports the work results to the common AUTOSAR model format. In addition, the AUTOSAR authoring tool would create a new catalog file that indicates which file contains the extended version of the `Implementation`.

Now the supplier submits catalog file in combination with model files back to the OEM. The latter takes the received catalog file and checks it for differences with the submitted file. Of course this only allows for checking of differences on file-level. However the OEM does only have to check the parts of the AUTOSAR model that is stored in changed files.

2.7 Filter and merge AUTOSAR models

[UC_IOAT_00009] Filter and merge AUTOSAR models ⌈ A filtered subset of an AUTOSAR model is passed to a supplier. The modified model needs to be merged back into the original model after being modified by the supplier.

The possible subsets are limited to the application of ≪`atpSplitable`≫.

If the model contains variants, it may not be possible that all variants can be bound before the merging, depending on the binding time. Hence, an AUTOSAR tool needs to be aware that the model that has been modified by the supplier contains variants that need to be bound at a later time. ⌋

2.8 Handling of identical double definitions

[UC_IOAT_00010] Handling of identical double definitions ⌈ When working on one particular component, there are `ARElement`s which need to be known but do not directly belong to the component. In this case the deliverable of the component development step may contain these objects such that it is "self contained".

By this, the component also documents how it was built. But in the integration step, this leads to duplicate elements which in fact are no duplicates. ⌋

When integrating such self contained components, these definitions may appear in the deliverable of all these components. As long as they are identical this is not a problem per se. But it violates the constraint that only the `atpSplitkey`s and their containers may be repeated in the different partial models. Nevertheless this use case needs to be handled properly.

This use case relates to `ARElement` such as `PortInterface`, `CompuMethod`, `SwBaseType`, `ApplicationDataType`, `ImplementationDataType`, `Unit`, `PhysicalDimension`, `DataConstr`, `PortPrototypeBlueprint`.

Please refer also to [UC_IOAT_00005].

3 Use-Case tracing (non-normative)

Use-Case	covered by
[UC_IOAT_00001] Support for subcontracting	The support for subcontracting is a very complex use case that is eventually affected by several places in this document. The primary prerequisite for subcontracting is the ability to exchange information as described in section 5.1. In some cases it might be necessary to also support concurrent modeling (e.g OEM provides supplier with updated model content). This aspect is covered by section 5.2. Finally, as models may be exchanged back and forth between OEM and suppliers the aspect of versioning (as described in section 5.7) could at least be of some importance. Also, the aspects described in section 5.3 are of importance for the implementation of the use case.
[UC_IOAT_00002] Support of different Versions of Meta-Model	The support for different versions of the Meta-Model is explained in section 5.6.
[UC_IOAT_00004] Concurrent Modeling	This use case is obviously directly covered by section 5.2.
[UC_IOAT_00005] Usage within the different steps of top-down functional development	This use case to some extent resembles [UC_IOAT_00001]. Therefore, the tracing to sections in this document is basically identical. In addition, however, the interoperability with specialized tools as described in section 5.5 might be of some interest for this use case.
[UC_IOAT_00006] Direct exchange of AUTOSAR model in tool-chain	As this uses case strongly emphasizes on the fact that AUTOSAR XML is the only format for information exchange of AUTOSAR model elements section 5.1 is obviously of paramount importance for the implementation of this use case. There are also relationships to section 5.3 because here the approach for grouping model artifacts is described. The information about the grouping can directly be used by the consuming tool.

Use-Case	covered by
[UC_IOAT_00008] Shipment of AUTOSAR Models and related Artifacts	This use case is directly covered by section 5.3.
[UC_IOAT_00009] Filter and merge AUTOSAR Models	This use case is primarily satisfied by section 5.1 and also section section 5.2 covers some aspects of this use case.
[UC_IOAT_00010] Handling of identical double definitions	This use case has a strong relationship to the aspects discussed in section 5.2.

4 Basic Concepts

[TR_IOAT_00071] Basic Concepts of Data Exchange ⌈ This chapter is intended to provide the reader with a more detailed insight into the exchange of information among different AUTOSAR tools. AUTOSAR has chosen XML as a language for exchange of data between different AUTOSAR tools. Therefore AUTOSAR tools SHALL be able to interpret and create AUTOSAR XML descriptions. The following section describes the different data representations which will be used in AUTOSAR and clarifies the relationship between the AUTOSAR meta-model, XML and the internal data structure of AUTOSAR tools. Additionally, common tasks that an AUTOSAR authoring tool needs to perform are defined. ⌋*(RS_IOAT_00001)*

4.1 Data representation

In a development process, many different tools with different representation of AUTOSAR models are used (Excel Sheets, Modeling Tools, UML, XML, etc.). Each tool and its underlying representation of data have their advantages and disadvantages. These tools and representations can be grouped into technological spaces.

A technological space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities [12]. Examples for technological spaces which are used within AUTOSAR are: meta-model, XML and AUTOSAR authoring tools (see Figure 4.1).

Technological spaces (e.g. meta-model and XML) are no islands. There are bridges between several technological spaces. The deliverable "AUTOSAR Interaction with Behavioral Models" explains how the AUTOSAR meta-model and the AUTOSAR concepts can be mapped to behavioral models and back again. The document "AUTOSAR model persistence rules for XML" [4] for example defines how to map an AUTOSAR meta-model to a W3C XML Schema.

Using XML and UML within AUTOSAR combines the strength of both technological spaces:

- AUTOSAR defined templates for data that is exchanged in AUTOSAR. Since XML is widely accepted as a standard for representation and exchange of structured data it was chosen to be the basis for the exchange of AUTOSAR models.
- Due to the complexity of the data and its interrelationships a manual creation of a consistent AUTOSAR XML schema turned out to be time-consuming and error prone. In addition the expressive power of XML schema is not sufficient for expressing content related constraints between data entities.
- Therefore a meta-model based approach was chosen to graphically describe the templates by means of UML2.0 class diagrams. Constraints that cannot be formulated graphically are described textually in the template specifications respectively as OCL (Object constraint language). The UML model which defines all

data entities and interrelationships that can be used for describing AUTOSAR systems and related artifacts is called AUTOSAR meta-model. An instance of the meta-model, i.e. a concrete description of software components, etc., is called an AUTOSAR model.

Figure 4.1 depicts the aforementioned technological spaces. The meta-levels (M0 to M4) show the correspondence of concepts in the different technological spaces. All concepts within one meta-level are strongly related.

Unlike the classical four-layer architecture used by OMG, five meta levels are shown. Starting at the lowest, most concrete meta level those are:

- **M0: AUTOSAR objects**
  This is the realization of an AUTOSAR system at work: For example a real ECUs executing a software image containing for instance the windshield wiper control software.

- **M1: AUTOSAR models**
  Models on this meta level are built by the AUTOSAR developers. They may define a software component called "windshield wiper" with a certain set of ports that is connected to another software component and so on. On this level all artifacts required to describe an AUTOSAR system are detailed, including re-usable types as well as specific instances of such types.

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

  Note that such an AUTOSAR model can be represented using various formats ranging from XML to C, even to PDF.

- **M2: AUTOSAR meta-model**
  On this meta level the vocabulary for AUTOSAR templates is defined. This vocabulary later can be used by developers of AUTOSAR based ECU systems.

  For example it is **defined on M2** that in AUTOSAR we have an entity called "software component" which among others aggregate an entity called "port". This definition ensures that the developer of an AUTOSAR software component can describe his particular component and its ports. This description is called an AUTOSAR model and **resides on M1**.

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

  Formally a template on M2 is still an instance of UML, but at the same time the template profile is applied, i.e. that additionally rules set out by the stereotypes in the profile need to be observed. The relevant details of the profile are specified in [3].

- **M4: Meta Object Facility**
  Just for completeness, OMG's MOF sits on the final metalevel M4. No further meta-levels are required since MOF is designed to be reflective.

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

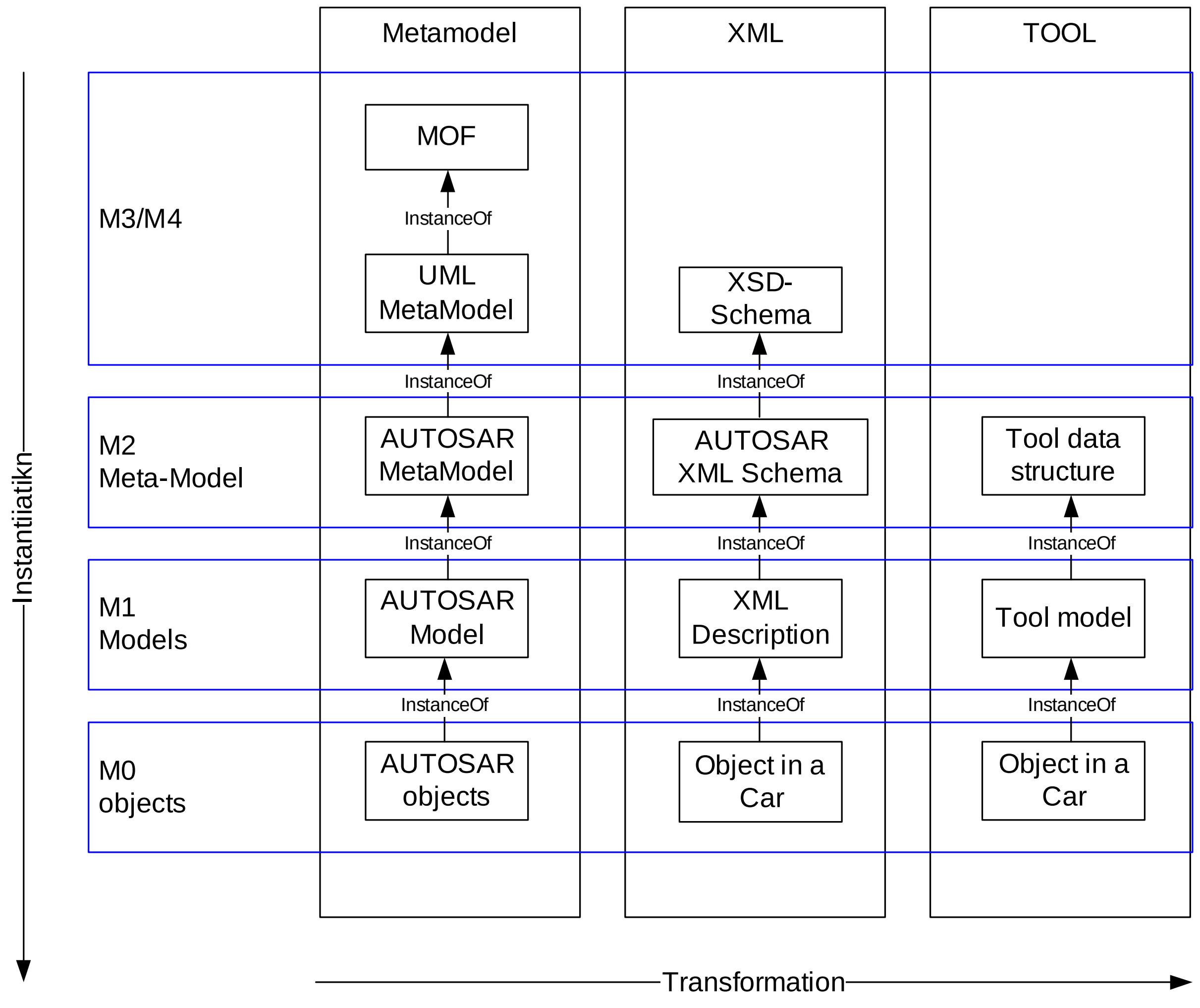

Figure 4.1: Technological spaces

4.1.1 Technological Space: "meta-model"

The Technological Space "meta-model" relates to the Model Driven Architecture (MDA) approach which was recently proposed by the OMG. According to MDA, the software development process is populated with a number of different models, each representing a particular view on the system being built. Models are written in the language of their meta-model[1].

The left part in Figure 4.1 shows the meta-model Technological Space as it is used in AUTOSAR.

- The lowest part called M0 corresponds to the real world. In the meta-model technological space AUTOSAR has no representation of M0 objects. It is mentioned for completeness only.
- All AUTOSAR models are at the level M1. For some standardized models, AUTOSAR uses an UML Object model.
- The M2 AUTOSAR meta-model is described by means of UML2.0 class diagrams and formally identified constraints in the AUTOSAR template specifications[2].
- A detailed description of the language that can be used for creating the AUTOSAR meta-model is given on the level M3 by the UML2.0 meta-model and the AUTOSAR Template Profile (see [3] for more information on the AUTOSAR Template Profile).
- The UML2 meta-model is defined by MOF which constitutes the level M4.

4.1.2 Technological Space: “XML”

Extensible Markup Language (XML) is a markup language standardized by W3C. It is widely accepted as a standard for representation and exchange of structured and semi structured data. The XML description is the central concept in the XML technological space. Descriptions are written in a syntax constrained by well-formedness and validity constraints. Well-formedness constraints are defined by the XML grammar rules, whereas the validity constraints are defined in a separate document called XML schema, which is written in a given schema language (W3C XML DTD [13], W3C XML Schema [14], etc.).

In other words: The XML grammar describes that a XML description contains opening and closing tags, etc. The XML schema defines e.g. which tags may be used in which combinations.

The middle part of Figure 4.1 illustrates the relations between an XML description, the XML grammar and a XML schema.

The technological space XML can be considered as a low level technological space: The AUTOSAR meta-model can be mapped to a XML schema [4]. But the original AUTOSAR meta-model cannot precisely be reconstructed out of the XML schema.

[1]AUTOSAR uses the database of the UML tool called "Enterprise Architect Database" to represent this technological space.

[2]these constraints may be specified as OCL constraints in future.

4.1.3 Technological Space: “Tool”

Each tool has its internal data structure which implements the concepts that can be used within the tool. This internal data structure is located at the meta-model level (M2) and defines the language which can be used to interpret or create descriptions or models (M1). The code that can be generated out of the model is again a different representation of the model (e.g. C). The runtime-instances of the code that are executed on an ECU in a car are represented on meta-level M0.

In most cases the internal data structure of an AUTOSAR tool is different from the structure defined by the AUTOSAR meta-model, e.g. for performance or historical reasons. In order to allow interoperability it is required to map the models represented by the internal data structure onto an AUTOSAR XML description. This mapping and the tool internal data structure is NOT subject of AUTOSAR standardization and therefore NOT in the scope of this document.

4.2 Abstraction levels of information exchange

Table 4.1 depicts several abstraction levels which will be used in this document for structuring the requirements on authoring tool interoperability and the AUTOSAR data exchange format. Each abstraction level is based on the underlying levels. For each abstraction level the mechanism that must be supported is described - beginning with the physical level (sets of files) until the semantic layer (semantic constraints formally specified in the AUTOSAR meta-model).

The abstraction levels "presentation level" and "application level" are not relevant for basic authoring tool interoperability but might have impact on the exchangeability of AUTOSAR authoring tools which perform a similar functionality (e.g.: if AUTOSAR authoring tools use a common graphical notation and provide similar mechanism for modifying AUTOSAR models, the effort for introducing another authoring tool is reduced).

In other words: each AUTOSAR authoring tool SHALL support the exchange of AUTOSAR models based on sets of XML-descriptions that can be distributed over several files. Each file in the set of files SHALL validate successfully against the AUTOSAR XML schema that is generated out of the AUTOSAR meta-model. The exchanged model SHOULD NOT violate semantic constraints. When exchanging AUTOSAR models an authoring tool needs to create an AUTOSAR XML description out of the internal data structure. Another authoring tool needs to interpret the AUTOSAR XML description and create its internal data-representation.

In addition to the common mechanism for exchanging AUTOSAR models, authoring tools can implement additional mechanisms. For example, an authoring tool could provide a plugin-interface which allows direct access to its internal data-structure. In this case a plugin and the authoring tool would be interoperable on the content level. However, even if an authoring tool supports additional mechanisms for exchanging AUTOSAR models at least the mechanisms defined in Table 4.1 SHALL be supported. Thereby AUTOSAR XML descriptions are the only standardized format.

Abstraction Level	Minimum supported mechanism for authoring tool interoperability	Document that describes further information
Application Level	e.g.: Advanced automatic features	Not specified by AUTOSAR
Presentation Level	Graphical notation	Partly specified by AUTOSAR: Graphical Notation [15]
Semantic Level	Semantic constraints precisely described in the meta-model (e.g.: criteria for compatibility of interfaces)	AUTOSAR meta-model [16], AUTOSAR Template UML Profile [3]
Content Level	Internal data-structure	This is subject to tool implementation and therefore not covered by AUTOSAR documents.
Data Format Level	AUTOSAR XML schema	Model Persistence Rules for XML [4]
Physical Level	Sets of files	This document

Table 4.1: Data Exchange Abstraction Levels

4.2.1 Physical level

The physical level defines the physical characteristics of the communication path. A physical representation could be one or more files or a data-stream.

4.2.2 Data format level

The data format level defines the format of the exchanged data. In AUTOSAR the data exchange format between different authoring tools is XML. The exchanged data SHALL be well-formed as defined in the W3C XML 1.1 Specification [13]. Additionally the exchanged XML descriptions SHALL be valid with respect to the AUTOSAR XML schema. We refer to this kind of data as "XML descriptions". This level can be implemented by off-the-shelf XML-parsers.

4.2.3 Content level

The content level defines the amount of information that can be exchanged. On the one hand it defines, which information is allowed to be given (boundedness). On the other hand, this level defines which information must at least be available (coverage). Authoring tools which base on this level can assume that at least a minimum and not more than a maximum of information is available. This level can be partly implemented by off-the-shelf validating XML-parsers which validate against a strict W3C XML schema (which can be generated out of the meta-model if the subset is formally defined). Some checks such as resolving of references must be implemented by the AUTOSAR authoring tool.

Example: An authoring tool might be specialized in the description of interfaces. This authoring tool might not understand any additional XML data. The content level needs to make sure that no data is transferred to the authoring tool which it does not understand.

Additionally the import and export of the AUTOSAR XML descriptions into the AUTOSAR authoring tool internal data structure is realized in this level. If the AUTOSAR XML description was split up over several files, the represented AUTOSAR model needs to be constructed. During this construction merge conflicts can occur and need to be resolved.

Example: references are resolved and potential conflicts such as multiple definitions of the same element are detected.

4.2.4 Semantic level

The definition of a standardized XML based exchange format is the first step towards successful interoperability of different authoring tools. The AUTOSAR XML data exchange format is defined by the AUTOSAR XML schema and therefore can only cover the "data format level" (validity according to the AUTOSAR XML schema).

In order to allow for seamless authoring tool interoperability all authoring tools must have the same interpretation of the semantics of the AUTOSAR models. For example all tools must have a common interpretation of the compatibility of instances of `PortInterface`s.

The validation of semantic constraints is not only mandatory for the exchange of AUTOSAR models - it is mandatory in any case (i.e. even if the information would not be exchanged among different tools) because it must be possible to check the consistency of AUTOSAR models during their creation and maintenance and before the model is exported to an AUTOSAR XML description.

The template specifications released by AUTOSAR [3][5][6][7][9][8] already contain discussions of particular semantic constraints, which basically implement the structure depicted in Figure 4.2.

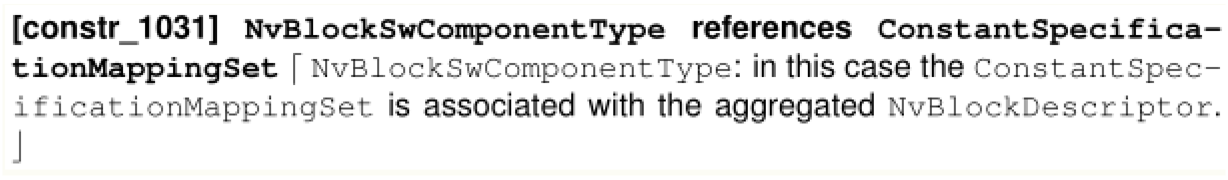
[constr_1031] `NvBlockSwComponentType` references `ConstantSpecificationMappingSet` ⌈`NvBlockSwComponentType`: in this case the `ConstantSpecificationMappingSet` is associated with the aggregated `NvBlockDescriptor`.
⌋

Figure 4.2: Example of a semantic constraint

4.2.5 Presentation level

In this level a more abstract access to the AUTOSAR model is supported. This can e.g. be realized by an API for a programming language, by a set of text tables and/or a graphical editor. This level does not support any automatic support for editing the AUTOSAR models. The API and representation of the model is highly depending on the AUTOSAR tool. Therefore AUTOSAR only specifies the graphical notation [15].

Example: editor with no automation support

4.2.6 Application level

The application supports automatic features such as algorithms for tool supported mapping of software components onto ECUs. This document does not specify the behavior of tools on this level, since the behavior is highly depending on the implementation of the tool. Additionally the "automatic" features have no impact on tool interoperability.

Example: semi-automatic algorithms for mapping signals onto buses.

5 Requirements on AUTOSAR Tools

5.1 Support for AUTOSAR XML data exchange

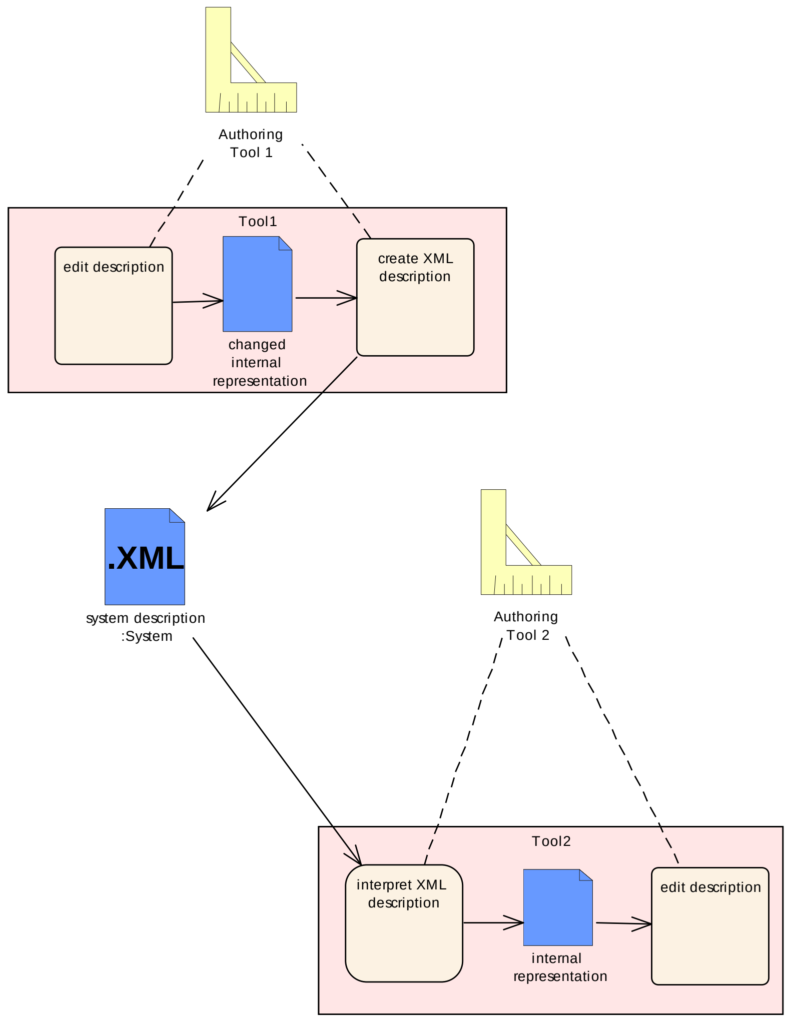

Figure 5.1: Support for AUTOSAR data exchange format

[TR_IOAT_00072] Support for AUTOSAR XML Data Exchange ⌈ When exchanging data between different departments or companies all involved parties need to agree on a common wording for the exchanged information. Otherwise all parties have a different understanding of the information. The same holds for exchanging AUTOSAR models between AUTOSAR authoring tools: Whenever AUTOSAR models are exchanged they need to be represented as AUTOSAR XML descriptions. ⌋*(RS_IOAT_00001)*

If the tools in a tool-chain are all able to deal with the full set of information as described in the AUTOSAR meta-model, they can perfectly exchange their AUTOSAR models. Of course this requires that all tools have implemented the functionalities which are defined in

- The physical level (e.g. they support files),

- the data format level (e.g. the files are valid with respect to the AUTOSAR XML schema),
- the content level (e.g. the data can be loaded in the tool internal data model), and
- the semantic level (e.g. ALL semantic constraints can be evaluated according to the template specifications).

Optionally these tools can use the same graphical notation as defined in the presentation level.

5.1.1 Physical level

5.1.1.1 AUTOSAR tool SHALL support sets of files

[TR_IOAT_00010] AUTOSAR tool SHALL support sets of files ⌈

Description:	AUTOSAR tools SHALL support for reading and writing single files and of sets of files that are stored in a file system. The tool SHALL provide a mechanism to select a specific file and sets of files in the file system.
Rationale:	An AUTOSAR XML description can be shipped in several files. Some files could contain data types others could contain interfaces, etc.
Use Case:	This allows the transport of models via CD, DVD, Email, etc. Splitting up an AUTOSAR model (represented as AUTOSAR XML descriptions) over several files supports concurrent modeling and a more fine-grained versioning. This allows development of parts of the model by different users or roles. Additionally it allows reuse of unchanged parts of the model.
Dependencies:	[TR_IOAT_00036], [TR_IOAT_00042], [TR_IOAT_00063]
Supporting Material:	ASAM Container Catalog as specified in [17]

⌋

The following details apply:

- An AUTOSAR tool SHALL be able to read the files in any order. Changing the order of reading SHALL not result in any change of the semantics of the model.
- An AUTOSAR authoring tool SHALL save changed model elements in the same file as they were read from.
- If the same element was read from two artifacts, it needs to be serialized back to these two ([TR_IOAT_00063]).
- An AUTOSAR authoring tool SHALL allow the user to specify in which file a newly created model element shall be saved.
- An AUTOSAR tool shall be able to read the files to be processed from an ASAM catalog file ([TR_IOAT_00036]).

5.1.2 Data format level

5.1.2.1 AUTOSAR tool SHALL support AUTOSAR XML descriptions

[TR_IOAT_00012] AUTOSAR tool SHALL support AUTOSAR XML descriptions ⌈

Description:	AUTOSAR tools SHALL support the interpretation and creation of AUTOSAR XML descriptions. These descriptions SHALL be "well-formed" and "valid" as defined by the XML recommendation, W3C XML 1.1 Specification [13], whether used with or without the document's corresponding AUTOSAR XML schema(s). In other words: Even if the tool does not use standard XML mechanisms for validating the XML descriptions it SHALL ensure that the XML descriptions can be successfully validated against the AUTOSAR XML schema.
Rationale:	Each AUTOSAR XML description file has confor to the AUTOSAR xml schema.
Use Case:	–
Dependencies:	A specialization of this requirement is defined in [TR_IOAT_00033].
Supporting Material:	W3C XML 1.1 Specification [13]

⌋

The following details apply:

- An AUTOSAR tool SHALL reject XML descriptions with violate XML's **well formedness** constraints as specified in [13]. Such files need to be fixed outside of AUTOSAR tools using a text editor. Proper error messages SHALL be provided, e.g. the messages from an XML parser).

- An AUTOSAR tool may accept XML description which do not validate properly against the XML schema. Nevertheless, for the processed subset all semantic constraints and also the schema constraints SHALL be evaluated. This applies in particular to specialized AUTOSAR tools which to not process the entire AUTOSAR model.

- An AUTOSAR tool may process an AUTOSAR description using a well-formed parser. Nevertheless, the result must exactly be the same as if the AUTOSAR description were parsed with validation.

- If an AUTOSAR tool wants to validate an AUTOSAR XML description against an AUTOSAR schema, it SHALL provide the necessary schema files in its own resources. An AUTOSAR tool shall use the `SYSTEM-Identifier` in the `xsi:schemaLocation` to identify an appropriate schema file.

  The `SYSTEM-Identifier` for the schema in the AUTOSAR XML description indicates the AUTOSAR schema version which was the basis for the **creation** of the description. It is very likely that a subsequent revision of the AUTOSAR schema can also be utilized to validate the description.

  Therefore an AUTOSAR tool may map the `SYSTEM-Identifier` for the schema to any resource which was declared backwards compatible by AUTOSAR. An AUTOSAR tool MUST NOT use the `SYSTEM-Identifier` directly as a filename.

AUTOSAR does not specify, **how** this mapping SHALL be performed, as there are various approaches to do this. As one example this may be done using SGML-OPEN-Catalog files [18]. In Example 5.1 two `SYSTEM-Identifer`s are mapped to the same schema file using an SGML-OPEN-Catalog-File.

Example 5.1

```
OVERRIDE YES
SYSTEM "AUTOSAR_4-0-0.XSD" "resources/AUTOSAR_4-0-1.xsd"
SYSTEM "AUTOSAR_4-0-1.XSD" "resources/AUTOSAR_4-0-1.xsd"
SYSTEM "http://www.w3.org/2001/03/xml.xsd" "resources/xml.xsd"
```

Example 5.2 illustrates the beginning of an AUTOSAR XML description with the requested attributes for AUTOSAR 4.0.x (see [TR_IOAT_00062] for more details):

Example 5.2

```
<?xml version="1.0" encoding="UTF8"?>
<AUTOSAR
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://autosar.org/schema/r4.0␣AUTOSAR_4-0-1.xsd"
 xmlns="http://autosar.org/schema/r4.0">
         ...
</AUTOSAR>
```

5.1.2.2 Authoring tool SHALL be able to import and export supported model elements as AUTOSAR XML descriptions

[TR_IOAT_00033] Authoring tool SHALL be able to import and export supported model elements as AUTOSAR XML descriptions ⌈

Description:	For all model elements that are defined by AUTOSAR and are supported by an authoring tool, the tool SHALL provide the user with the possibility to import them from XML descriptions and export them as XML descriptions that validate successfully against the AUTOSAR XML schema. Even if the tool has non AUTOSAR exchange possibility of models it SHALL nonetheless support the creation of the corresponding AUTOSAR XML description.
Rationale:	It is possible that between two tools or independent components of the same tool exists some data exchange mechanism. Such mechanism might make it possible for the tool to bypass AUTOSAR XML descriptions even if the model can be represented by AUTOSAR XML descriptions.
Use Case:	Avoid proprietary exchange formats that bypass the standardized AUTOSAR XML descriptions and therefore endanger the interoperability of tools from different vendors.
Dependencies:	[TR_IOAT_00012]
Supporting Material:	–

⌋

5.1.2.3 Authoring tool SHALL support well defined serialization

[TR_IOAT_00062] Authoring tool SHALL support well defined serialization ⌈

Description:	An AUTOSAR authoring tool shall provide a serialization for XML as shown in the table below, in particular in Table 5.1
Rationale:	In order to support a direct comparison of AUTOSAR XML descriptions with a text comparison tool it is essential that the XML is generated in a reliable and standardized manner.
Dependencies:	[TR_IOAT_00012]
Supporting Material:	–

⌋*(UC_IOAT_00004, UC_IOAT_00008)*

In order to support a direct comparison of AUTOSAR XML descriptions with a text comparison tool it is essential that the XML is generated in a reliable and standardized manner.

This requirement also supports a well defined validation against an xml schema.

Therefore an AUTOSAR tool shall support serialization as follows:

- In AUTOSAR XML the root element shall contain the attribute `xsi:schemaLocation`. This attribute is list of URI/URL pairs which associates a SYSTEM-Identifier (the url) to a name space identifier (the URI):

  ```
  <AUTOSAR xsi:schemaLocation="{AUTOSAR␣Namespace}␣{SYSTEM-Identifier}">
  ```

  AUTOSAR XML description SHALL serialize the root element as follows (see also example 5.2:

  ```
  <?xml version="1.0" encoding="UTF8"?>
  <AUTOSAR
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="{AUTOSAR␣Namespace}␣{SYSTEM-Identifier}"
   xmlns="{AUTOSAR␣Namespace}">
          ...
  </AUTOSAR>
  ```

- The `SYSTEM-Identifier` for the schema in the AUTOSAR XML description indicates the AUTOSAR schema version which is the basis for the **creation** of the description. It is very likely that a subsequent revision of the AUTOSAR schema can also be utilized to validate the description. Therefore an AUTOSAR tool may map the SYSTEM identifier for the schema to any resource which was declared backwards compatible by AUTOSAR.

  The `SYSTEM-Identifier` in the AUTOSAR XML description shall match the following pattern:

```
AUTOSAR_{major}-{minor}-{revision}.xsd
```

In particular no path shall be part of the `SYSTEM-Identifier`.

- AUTOSAR XML descriptions SHALL use the file extension ".arxml" (short for AUTOSAR XML).
- Unless the meta model flags multiple elements as `{ordered}` the elements with upper multiplicity $>$ 1 shall be serialized in a predictable order. In particular, the order shall be defined by the expression mentioned in the `atp.splitKey`. If no splitkey is defined the order shall be defined by the contents of `shortName`, `shortLabel`.
- XML comments may be silently ignored and need not to be serialized again. XML comments are not treated as part of the AUTOSAR model.
- XML processing instructions may be silently ignored and need not to be serialized again. It is allowed that an AUTOSAR tool places processing instructions to the AUTOSAR XML description for specific purposes.
- primitives such as `Numerical` etc. SHALL be serialized as read from the AUTOSAR XML description respectively as entered by the user in an AUTOSAR Authoring tool.
- empty elements (if there are those) should be serialized as start/end tag, not as ’emptytag’.

  In particular

  `<VALUE></VALUE>` instead of `<VALUE/>` for primitives

  respectively for empty containers

  ```
  <ELEMENTS>
  </ELEMENTS>
  ```
- The XML description SHALL be formatted as shown in Table 5.1:

Applied to	Strategy	description
default approach	*NewLine*: Element is a block of its own	NewLine means in particular: • indentation shall be 2 characters per level • the start tag of the element shall be on a new line • the XML attributes shall be sorted alphabetically. If more than one XML attribute, each one shall be on its own line • the start shall be indented according to the nesting level of XML tag • the end tag shall be on a new line and indented like the start tag • no newline shall be placed after the end tag, this is inserted by the next tag if necessary • the content will be indented one step more than the start tag
`Identifiable`s	*BlankLine*: Element is a visible block	the approach is as *NeweLine* but with a blank line before the block

Applied to	Strategy	description

shortName and shortLabel	*KeepLine*: element remains on the curent line	Keepline is in particular:
		• The start tag of the element will not start on a new line
		• the position in the parent element will be kept as is
		• no newline will be placed after the end tag, this is inserted by the next tag if necessary
		`<UNIT><SHORT-NAME>Perc</SHORT-NAME>`
		`  <DESC>`
		`    ...`

Primitives (either modeled as UML-attribute or as aggregation of a primitive	*OneLine* Element is displayed in one line	The element starts on a new line, is indented and presented in one line. The end tag is in the same line as the start tag and the content of the element.

Properties of ≪atpMixedString≫	*InLine*: Element is floating within text	Surrounding whitespace of the element is not changed. No new line is inserted before or after the tags. Whitespace within the element is also not changed. In the following example the element `<E>` is formatted according to the *InLine* approach.
		`<L-1 L="EN">This`
		`    is  <E>bold  </E>  style  </L-1>`

VerbatimString elements with xml:space set to preserve	*keepWhitespace*	White space in the element is kept as is.
elements with with no xml:space or set to default	*normalizeWhitespace*	Normalize whitespace includes: • leading and trailing whitespace will be removed • consecutive white spaces will be replaced by a single blank • no wrapping will be performed • carriage returns will be replaced by blank • child(inline)-elements are treated as one non whitespace character

Table 5.1: Approaches for formating XML serialization

The following example 5.3 illustrates these approaches:

Example 5.3

```
<UNIT><SHORT-NAME>Perc</SHORT-NAME>                    <!-- KeepLine -->
  <DESC>                                               <!-- NewLine -->
    <L-2 L="EN">a percentage...</L-2>                  <!-- OneLine -->
  </DESC>
  <DISPLAY-NAME>%</DISPLAY-NAME>                       <!-- OneLine -->
</UNIT>
                                                       <!-- BlankLine -->
<UNIT><SHORT-NAME>PercPerSec</SHORT-NAME>              <!-- KeepLine -->
  <DESC>                                               <!-- NewLine -->
    <L-2 L="EN">time-derivative of percent</L-2>       <!-- NewLine
                                                            NormalizeWhitespace

                                                       -->
  </DESC>
  <DISPLAY-NAME>%/s</DISPLAY-NAME>                    <!-- OneLine -->
</UNIT>
```

5.1.3 Content level

5.1.3.1 Authoring tool SHALL NOT change model contents without the intention of the user

[TR_IOAT_00007] Authoring tool SHALL NOT change model contents without the intention of the user ⌈

Description:	An AUTOSAR authoring tool SHALL NOT perform any changes on the AUTOSAR model while interpreting and creating AUTOSAR XML descriptions. If the user doesn't explicitly trigger or confirm any changes then the semantics of the AUTOSAR model represented by the original XML description SHALL be equivalent to the semantics of the model represented by the created XML description. This includes in particular • that an authoring tool SHALL preserve references even if the target was not available in the input XML description • that the format of primitives such as `Numerical`, `Integer` is not changed • that the package structure of the model is not changed • that the `base` attribute in references is not changed • that uuids are not allowed to be removed or changed
Rationale:	–
Use Case:	The meta-model contains some elements that are marked by ordered. The order in the XML representation may not change without the intention of the user when interpreting and creating the XML description. Other use-cases may apply as well.
Dependencies:	[TR_IOAT_00024]
Supporting Material:	–

⌋

5.1.3.2 Authoring tool SHALL support exchange of partial information

[TR_IOAT_00035] Authoring tool SHALL support exchange of partial information ⌈

Description:	An AUTOSAR authoring tool SHALL support the exchange of AUTOSAR models that are not complete.
Rationale:	It SHALL be possible to exchange intermediate work products.

Use Case:	[UC_IOAT_00001], [UC_IOAT_00005]
Dependencies:	[TR_IOAT_00024]
Supporting Material:	–

⌋*(UC_IOAT_00001, UC_IOAT_00005)*

5.1.3.3 Authoring tool SHALL support AUTOSAR extension mechanism

[TR_IOAT_00048] Authoring tool SHALL support AUTOSAR extension mechanism ⌈

Description:	An AUTOSAR authoring tool MAY support the AUTOSAR extension mechanism if applicable. Tools that do not need the additional information for its intended purpose SHALL ignore that information. If several extensions are defined then the tool SHOULD support the relevant extensions and SHALL ignore the irrelevant extensions.
Rationale:	For some use-cases it is required to exchange information that is not (yet) captured in the standard data exchange format. This additional information is often specific to a tool (e.g. allow for round-trip engineering) or a tool-chain within a development process.
Use Case:	Using AUTOSAR descriptions in specific processes.
Dependencies:	–
Supporting Material:	–

⌋

5.1.3.4 Authoring tool SHOULD maintain references

[TR_IOAT_00067] AUTOSAR Authoring tool SHOULD maintain references ⌈

Description:	An AUTOSAR authoring tool SHOULD provide means to maintain references including `referenceBase` and attribute `base`
Rationale:	The implementation of references is part of the AUTOSAR model and shall be handled properly.
Use Case:	[UC_IOAT_00004]
Dependencies:	[TR_IOAT_00007], [TR_IOAT_0003], [TR_IOAT_00064]
Supporting Material:	–

⌋*(UC_IOAT_00004)*

5.1.3.5 Authoring tool SHOULD follow specified access rights

[TR_IOAT_00074] AUTOSAR Authoring tool SHOULD follow specified access rights ⌈

Description:	An AUTOSAR authoring tool SHOULD provide an option to follow access rights. If this option is selected, it SHALL follow the access rights specified according to [TPS_GST_00226]
Rationale:	The agreed roles and their permissions need to be supported by a tool.
Use Case:	[UC_IOAT_00004]
Dependencies:	[TR_IOAT_00007], [TR_IOAT_0003], [TR_IOAT_00043]
Supporting Material:	–

⌋*(UC_IOAT_00004)*

5.1.4 Semantic level

5.1.4.1 Authoring tool SHALL support validity checks

[TR_IOAT_0003] Authoring tool SHALL support validity checks ⌈

Description:	An AUTOSAR authoring tool SHALL support validating the consistency of the AUTOSAR model to the AUTOSAR meta-model, including the semantic constraints. The validation SHALL be performed when interpreting and creating AUTOSAR models. A tool SHOULD allow the user to trigger validity checks manually. The tool SHALL allow for interpreting and creating AUTOSAR XML-descriptions that are not valid with respect to the semantic constraints. If the tool is not able to interpret the file due to the violation of some semantic constraints it SHALL notify the user and indicate the location of the violating elements. The SHALL perform semantic checks according to the intended task. The Tool shall not produce errornous results from semantically incorrect input without notice to the user.
Rationale:	–
Use Case:	Support to create models of appropriate quality. User wants to correct semantical problems.
Dependencies:	Requirement on the Resolution of references in [TR_IOAT_00064]. The violation of semantic constraints shall be reported according to [TR_IOAT_00065]. Validity checks shall be performed after loading all partial models (see [TR_IOAT_00042].
Supporting Material:	–

⌋

5.1.4.2 AUTOSAR tool SHALL support variants

[TR_IOAT_00060] AUTOSAR tool SHALL support variants ⌈

Description:	Starting with AUTOSAR 4.0, AUTOSAR models may contain variation points and tables of evaluated variants. An AUTOSAR tool needs to understand the semantics of variation points and handle variability correctly.
Rationale:	An AUTOSAR model which contains variants is subject to relaxed constraints on multiplicity and uniqueness of `shortName`s. Hence, the model is no longer backward compatible, and a tool which does not understand variant handling cannot interpret such a model correctly; it is not sufficient to just ignore the variation point information.
Use Case:	Party 1 sends a model which contains variants to party 2, which generates code from this model, but does not resolve all variants. Party 1 gets back a result with some variants bound, and other variants left “open” for binding at compile time or later.
Dependencies:	–
Supporting Material:	Variant handling is described in Chapter 7 in the Generic Structure Template [3].

⌋

5.1.5 Presentation level

There are no general requirements on AUTOSAR Tools presentation level. Please refer to chapter 5.8 “Standardized error handling” which also relates to the presentation level.

5.2 Support for concurrent modeling

During the development of AUTOSAR systems, models will be passed between different parties (e.g. from OEM to supplier and back). The fact that it is not possible or wanted for all involved parties to work on the same repository, those parties will work on different instances of the same model, e.g. enrich the models, add new elements, implement the model etc. At some point in time, these models will be merged into one complete model for the entire system: The consistency of the merged model must be ensured and conflicts need to be resolved.

Since AUTOSAR models can be stored as AUTOSAR XML descriptions in several files (which could be modified independently), an AUTOSAR authoring tool must support merging the models stored in these files into a consistent internal data representation. This includes providing a mechanism for manually or rule-based resolving of merge conflicts.

Figure 5.2 shows the concept of handling AUTOSAR models which can be split up over several files. The AUTOSAR authoring tool needs to interpret the contents stored in each single file (partial model) and needs to merge them into the tool internal data structure. After having finished modifying the model an XML description is created which can again be split up over several files (the granularity of information that can be split up over several files is defined in [TR_IOAT_00038]). Note that the merging shall not be performed on the XML text level. It shall be performed on the level of the partial models represented by XML.

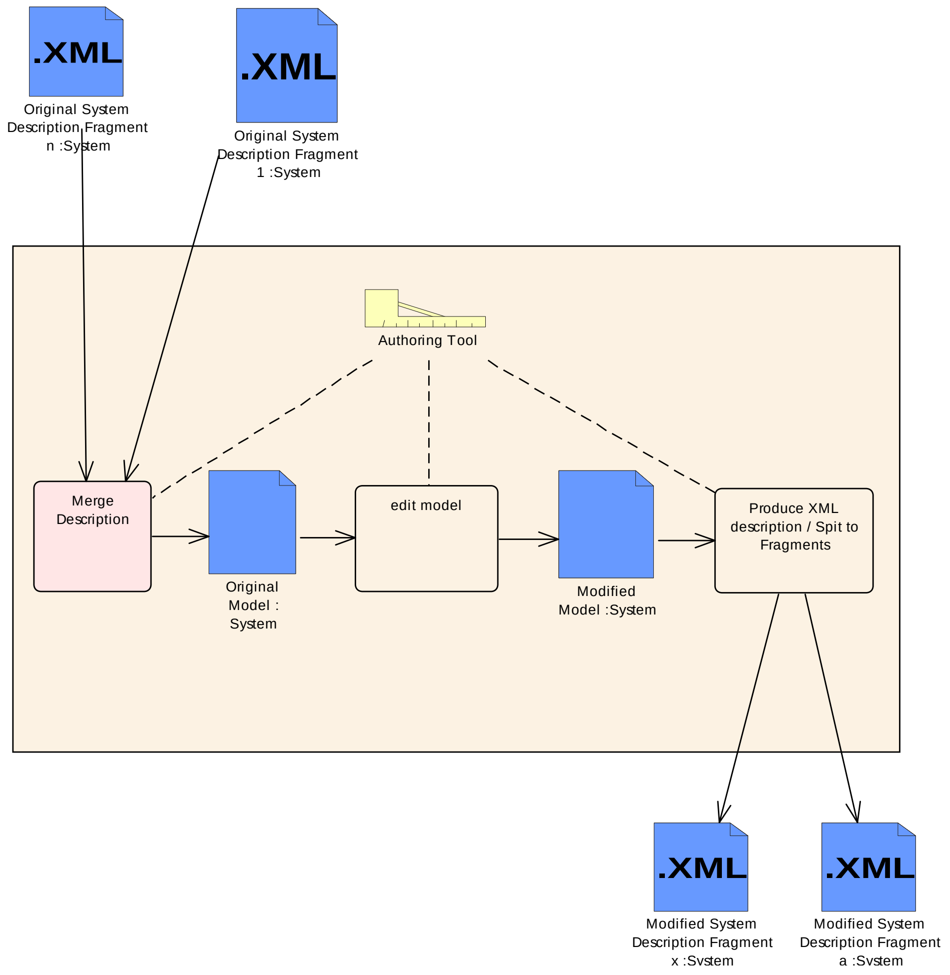
.XML
Original System Description Fragment n :System
.XML
Original System Description Fragment 1 :System
Authoring Tool
Merge Description
Original Model : System
edit model
Modified Model :System
Produce XML description / Spit to Fragments
.XML
Modified System Description Fragment x :System
.XML
Modified System Description Fragment a :System

Figure 5.2: Concept of handling AUTOSAR models which can be split up over several files

5.2.1 Detection of differences between models

5.2.1.1 Authoring tool SHOULD provide a mechanism for showing differences between AUTOSAR models

[TR_IOAT_00043] Authoring tool SHOULD provide a mechanism for showing differences between AUTOSAR models ⌈

Description:	An AUTOSAR authoring tool SHOULD provide a mechanism for showing differences between AUTOSAR models. These differences could be represented in textual or in graphical ways.
Rationale:	Identification of differences between two versions of an AUTOSAR model.

Use Case:	• OEM wants to find out which parts of the model have been modified by the supplier. • The user wants to check if a merge of two models was executed as expected.
Dependencies:	–
Supporting Material:	–

⌋

5.2.1.2 Definition of differences

The reconciliation of two independently modified models involves several activities.

1. Identification of model elements which are only available in one model.
2. Identification of model elements that are available in both models and are identical in both models. Two model elements are considered identical if
   (a) their attributes values are equal,
   (b) their short-name references to other elements are equal and
   (c) they are composed of identical elements (recursiveness).
   (d) for aggregations with upper multiplicity greater than one, and specified as `{ordered}`, the elements appear in the same sequence
   (e) for aggregations with upper multiplicity greater than one and **not** specified as `{ordered}` both sets contain the same elements
3. Identification of model elements that are available in both models and are different in the models.

The activities 2 and 3 require a precise definition of the concept of a "changed" model element. In the following section we refer to meta classes that are specializations of meta-class `Identifiable` as identifiables. An identifiable is considered as "changed" if any of the following is changed:

- Any of its attributes of simple datatype changed its value.
- The identifiable or any parts (recursive) that are not identifiables themselves are modified. So the granularity of possible detection of changes is the identifiable. The rationale for this is that the Identifiable is considered as having an identity of its own for which the difference is indicated separately.

The following section explains this definition in more details:

5.2.1.3 Definition of differences - aggregation

Figure 5.3 illustrates the granularity of modification detection: “A” is a specialization of `Identifiable`. “B” is aggregated by “A” (“B” is a part of “A”) and “B1” is aggregated by “B” (“B1” is part of “B”). “A” is considered changed (indicated by dark color) if the underlying structure or any of the structure’s elements change. However it depends on the particular use case if nested changes shall be indicated at higher levels to the user.

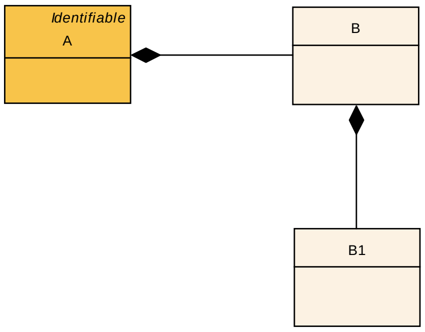

Figure 5.3: Modification detection - aggregated meta classes that are not identifiable

Difference on aggregations marked as ≪`atp.splitable`≫ require a more elaborate consideration depending if partial or merged models are compared:

1. When comparing a merged model the rule of aggregations apply as mentioned above.
2. When comparing a partial model, only the partial model which contains the change also indicates the difference. In figure 5.4 the change of “C” is only indicated in the “Lower Partial Model”. Reason is, that the “Upper Partial Model” is not aware that there is another part.

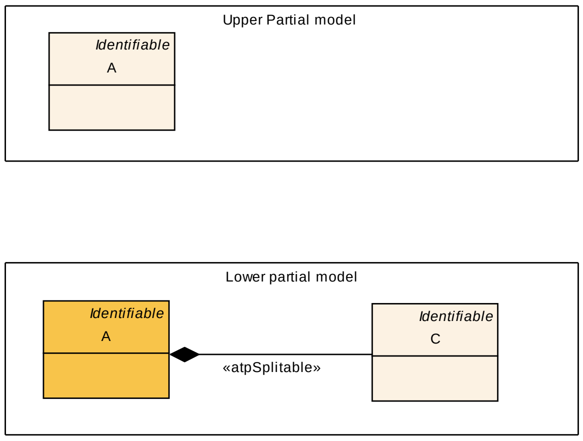

Figure 5.4: Modification detection - meta classes aggregated as ≪`atp.splitable`≫

5.2.1.4 Definition of differences - references

If an Identifiable "A" has references to other Identifiables (see Figure 5.5), "A" is considered changed, if a new reference is added/deleted or the string that internally represents the reference is changed. "A" is not considered changed, if the referenced element changes. Please note that the AUTOSAR references are based on the shortName as explained in the "Generic Structure Template" [3] and therefore changing the shortName of an instance of "C" implies updating all references to the instance. In this special case "A" is considered changed if the shortName of "C" is updated (the string representing the reference needed to be updated too).

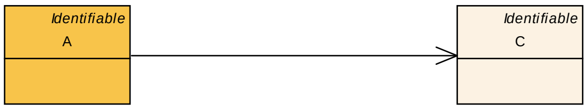

Figure 5.5: Modification detection - aggregated meta classes that are identifiable

5.2.1.5 Algorithm for comparison of model elements

The concrete algorithm of comparison is left open to the implementation, but it is required that it detects all of the changes described above. Possible implementation could be a comparison on a per-attribute basis or the calculation and comparison of a signature value (e.g. calculated by the MD5 algorithm) for any Identifiable. Since somebody might change the content of an XML description with an XML editor without updating the signature values, an authoring tool should not rely on the correctness of the checksums that are transmitted with an XML description. However, if a user can ensure that the checksum values fit to the content then they can be used in order to increase the performance of the comparison.

5.2.1.6 Authoring tool SHALL support unique identification of model elements

[TR_IOAT_00024] Authoring tool SHALL support unique identification of model elements ⌈

Description:	• Upon user request, an AUTOSAR authoring tool MAY create XML descriptions which contains uuids for each Identifiable. • An AUTOSAR authoring tool SHALL continue interpreting an AUTOSAR XML description even if uuids are missing. • An AUTOSAR authoring tool SHALL not change the uuid while processing an AUTOSAR model unless it is explicitly intended by the user.
Rationale:	Ability to trace model elements independently of content modifications.

Use Case:	Export of data from a database with unique uuids, change data in external tools and then re-import into the database. During the re-import the uuids aid the unambiguous assignment of model elements to data in the database.
Dependencies:	[TR_IOAT_00042], [TR_IOAT_00035], [TR_IOAT_00007]
Supporting Material:	–

⌋

5.2.1.7 Examples of differences between models (non normative)

The following figures show an original and a changed model. The model elements that are marked in green indicate which element is considered to be changed: Figure 5.6 shows the original model. The SenderReceiverInterface "interface" and the ApplicationPrimitiveDataType "velocityType" are defined within the ARPackage "OEM". The SenderReceiverInterface "interface" has a VariableDataPrototype "velocity" which is of type "velocityType". The invalid Value[1] of the ApplicationPrimitiveDataType "velocityType" is set to 10000.

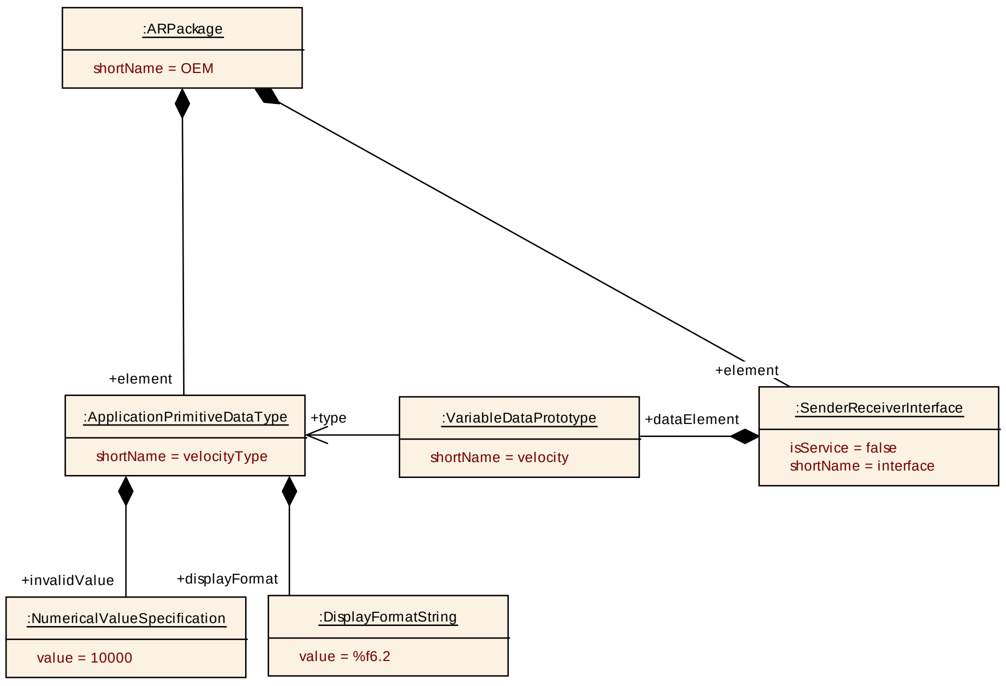

Figure 5.6: Original Model

The model which is described in Figure 5.7 contains two additional model elements: VariableDataPrototype "buttonPressed" and the ApplicationPrimitiveDataType "Boolean". These elements are marked as changed because they were added. The ARPackage "OEM" and the SenderReceiverInterface "inter-

[1]Note that ValueSpecification is simplified in this diagram

face" are marked as changed because aggregations to the new model elements have been added.

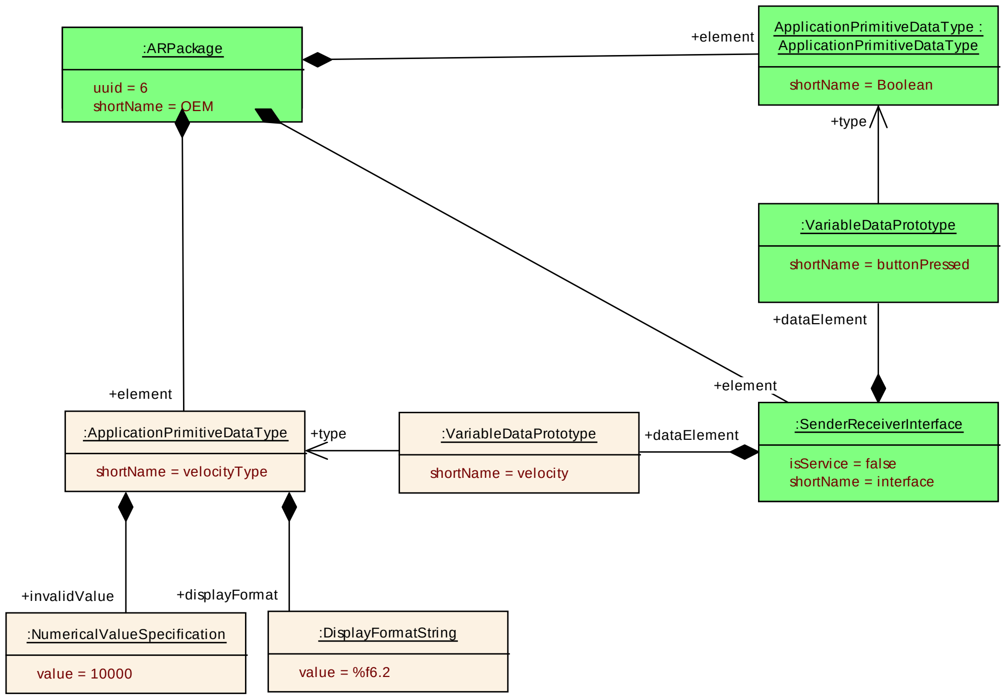

Figure 5.7: Modified model - added Data Element and Data Types

In Figure 5.8 the invalid value of ApplicationPrimitiveDataType "velocityType" was extended. The change in the object ValueSpecification is propagated to the Identifiable ApplicationPrimitiveDataType.

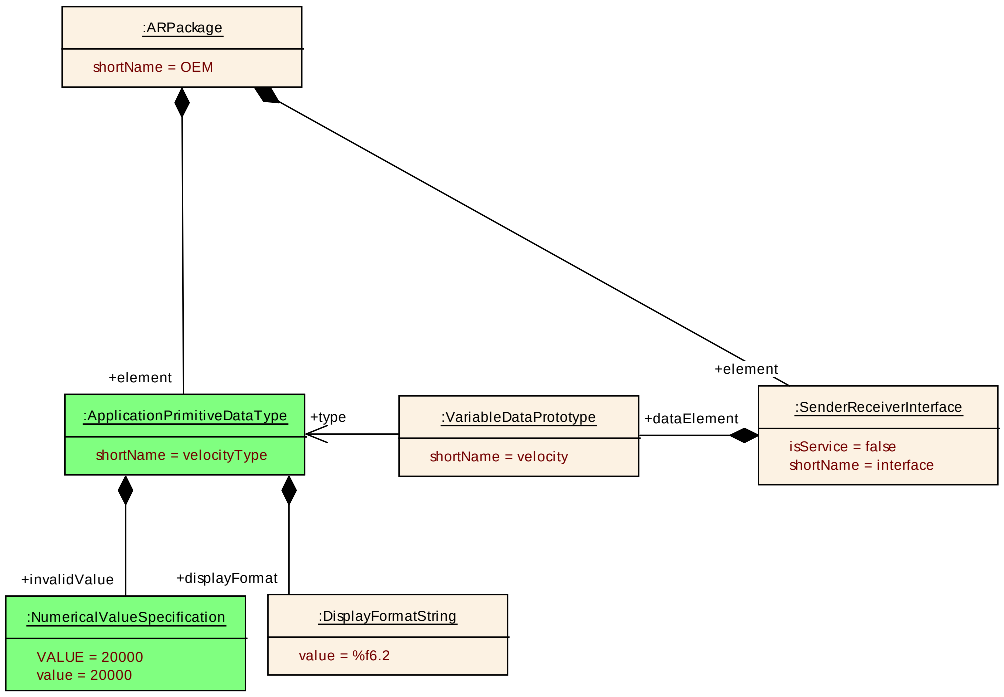

Figure 5.8: Modified model - changed Invalid Value

The shortName of the ApplicationPrimitiveDataType in Figure 5.9 was changed from "velocityType" to "speedType". Changing the shortName of an object implies a change of the references to this object. Therefore the VariableDataPrototype "velocityType" is also changed. All these changes are propagated to the containing ARPackage "OEM"

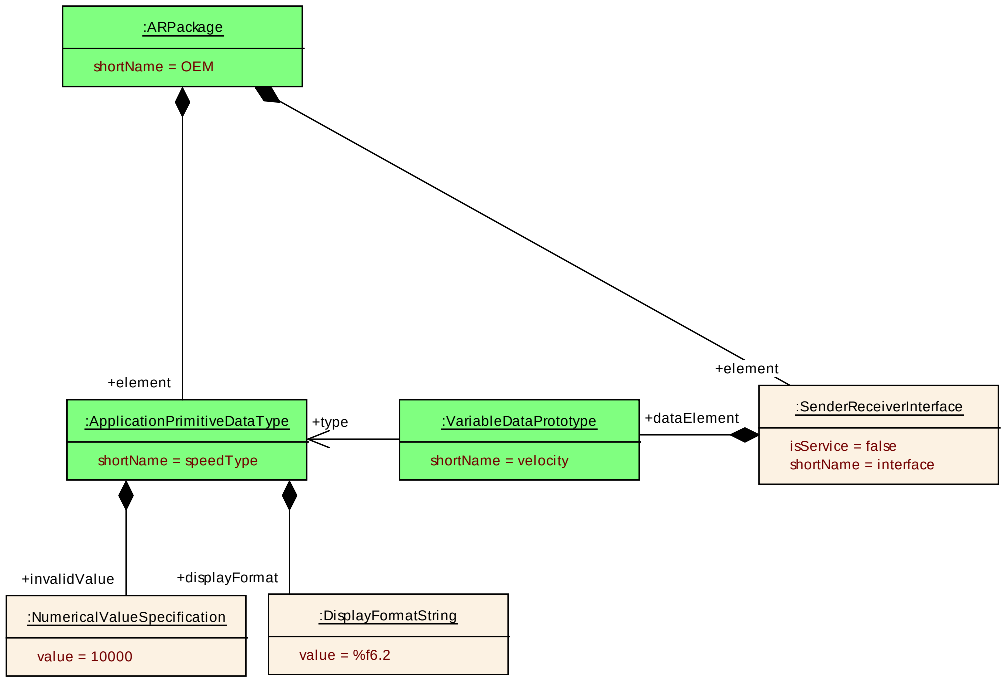

Figure 5.9: Modified model - renamed Element (“speedType”)

If a model element is moved to a different ARPackage then only the source and the target ARPackage are changed (Their list of elements changes). This is described in Figure 5.10. Please note that all absolute references that point to moved elements need to be changed as well. This is required because AUTOSAR uses absolute short name paths for referencing elements. If references would have been relative references then the references would not necessarily change depending on the layout of the referenceBases.

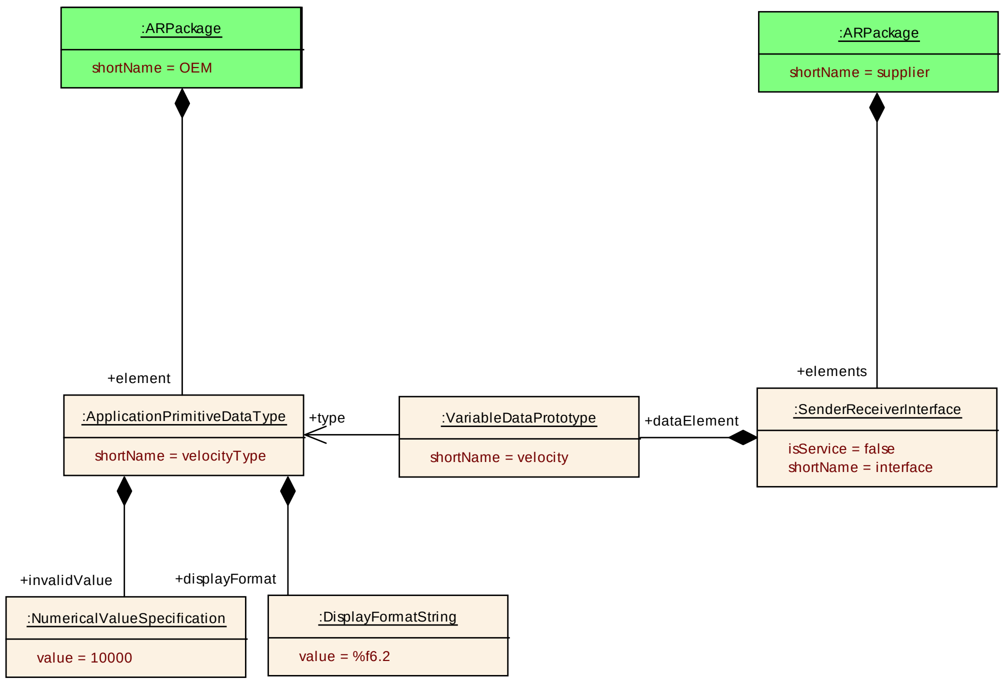

Figure 5.10: Modified model - moved Element ("interface") to another Package

5.2.2 Merging models

The following sections describe requirements on interpreting AUTOSAR models that have been split up into several partial-models, each stored in an individual file. In order to create an overall model the content described in the partial-models need to be merged. The following algorithm describes a 2-way merge. Due to missing experience with handling complex AUTOSAR models in development processes, strategies for resolving merge conflicts are left open to implementation of tools.

In order to reduce manual interaction for resolving merge conflicts, tools may take additional information into consideration. This information can e.g. be derived out of an earlier version of the overall model (origin model, 3-way merge) or out of information stored in meta-data for data exchange (see section 5.9).

5.2.2.1 Authoring tool SHALL be able to handle partial AUTOSAR models

[TR_IOAT_00061] Authoring tool SHALL be able to handle partial AUTOSAR models ⌈

Description:	AUTOSAR authoring tools SHALL support handling of partial models. This includes in particular support of dangling references as well as appropriate constraint validation. When storing an AUTOSAR partial model as an AUTOSAR XML description this needs to be valid with respect to the AUTOSAR XML schema: i.e. each of such files contains a full path beginning from the root element AUTOSAR.

Rationale:	Allow for splitting up AUTOSAR models over several sub models which can be stored, versioned, and modified independently.
Use Case:	According to various process approaches, an AUTOSAR model may be split up into several files and have to be merged to a complete and consistent model.
Dependencies:	–
Supporting Material:	See "Generic Structure Template" [3] for detailed description on how to mark an aggregation as ≪atpSplitable≫.

⌋

5.2.2.2 Authoring tool SHALL support the merging of AUTOSAR models

[TR_IOAT_00042] Authoring tool SHALL support the merging of AUTOSAR models ⌈

Description:	An AUTOSAR authoring tools SHALL support the merging of AUTOSAR models that have been split up and stored in multiple partial models. The minimum granularity of an AUTOSAR model is explicitly modeled in the AUTOSAR meta-model: If an aggregation is marked as ≪atpSplitable≫, then the aggregated elements MAY be described in different files. If the aggregation is not marked as ≪atpSplitable≫, then the aggregated content SHALL be stored in the same file as the aggregating element. Merging of a model also includes the resolution of references. The tool SHALL be able to read the submodels in any order. There is no preference.
Rationale:	Allow for splitting up AUTOSAR models over several sub models which can be stored, versioned, and developed independently.
Use Case:	When storing an AUTOSAR model as an AUTOSAR XML description in several files, each file needs to be valid with respect to the AUTOSAR XML schema: i.e. each file contains a full path beginning from the root element AUTOSAR. Let's assume that an AUTOSAR model defines an AtomicSwComponentType "A" and a SenderReceiverInterface "S" within the same ARPackage named "pkg". The XML description of "A" can be located in another file than the XML description of "S". Each file would contain the description of the ARPackage "pkg". While interpreting the contents of the two files the AUTOSAR authoring tool must make sure that only one instance of "pkg" is created in the internal representation of the AUTOSAR model.
Dependencies:	This requirement is refined by [TR_IOAT_00044], [TR_IOAT_00043], [TR_IOAT_00040], [TR_IOAT_00063], [TR_IOAT_00064] and [TR_IOAT_00024].
Supporting Material:	See "Generic Structure Template" [3] for detailed description on how to mark an aggregation as ≪atpSplitable≫.

⌋

5.2.2.3 AUTOSAR tool SHALL resolve references

[TR_IOAT_00064] AUTOSAR authoring/integration tools SHALL resolve references ⌈

Description:	When an AUTOSAR tools loads partial models, it shall be able to handle references. This includes: • late binding of references: When a partial model is loaded which contains `Identifiable`s which are referenced by previously loaded artifacts, the references shall be bound • handling dangling references: if a reference cannot be bound, it shall be flagged as error, but the partial model shall still be loaded. Open references shall be visible in the authoring systems (e.g. the name of a referenced object shall be shown even if the reference cannot be resolved).
Rationale:	Allow for integration of components that have been developed or modified by different parties but referring common definitions which are not yet available.
Use Case:	[UC_IOAT_00010], [UC_IOAT_00001]
Dependencies:	[TR_IOAT_00067]
Supporting Material:	–

⌋*(UC_IOAT_00010, UC_IOAT_00001)*

5.2.2.4 Handling Conflicts

In case of conflicts, these need to be resolved with human interaction, since semantic knowledge is needed in most cases for the decisions. Basically, for handling conflicts, there is a need for regulation for mastership and how this is to be applied.

Merging tools might infer suggestions from additional information. If the sub-models are derived from a common origin, the 3-way merge algorithms can be applied. Additionally meta-data for data exchange could be used. E.g.: A timestamp could be used to indicate the time of modification of an element (the initial creation of an instance is considered as a modification, too), where modification means any change as described in the section above. Timestamp is an optional field, since its existence is not critical for the process.

5.2.2.5 AUTOSAR tools SHALL accept double defined `ARElement`s as long as their nonSplitables are the same

[TR_IOAT_00063] AUTOSAR tools SHALL accept double defined `ARElement`s as long as their nonSplitables are the same ⌈

Description:	An AUTOSAR tool shall keep track of the artifacts where "duplicate `ARElement`s" came from. An AUTOSAR tool shall indicate double defined `ARElement`s with different nonSplitables as error. All related artifacts shall be indicated to the user. There is no preference rule. This duplicate handling is limited to `ARElement`s which are not split into partial models and therefore loaded from one artifact.
Rationale:	Allow for integration of components have been modified by different parties but referring common definitions.
Use Case:	[UC_IOAT_00010], [UC_IOAT_00001]
Dependencies:	–
Supporting Material:	–

⌋*(UC_IOAT_00010, UC_IOAT_00001)*

5.2.2.6 Handling merge conflicts: optimistic approach

In the case of an optimistic approach for resolving conflicts, work is allowed on copies of a model and only when the models are synchronized/integrated potential conflicts are be resolved. Whenever a merge conflict is detected (i.e. two AUTOSAR models contain model elements which have the same `shortName` but a different description) the conflict needs to be resolved. This could be implemented by the tool by interactively asking the user to choose the right model element.

Additional meta-data on the model elements can help finding out the correct model element. E.g. if a timestamp is assigned to a model element it can be decided which information is newer than the other. Another approach for resolving or even avoiding merge conflicts is described in section 5.2.2.8.

5.2.2.7 Authoring tool SHOULD provide a mechanism for resolving merging conflicts

[TR_IOAT_00044] Authoring tool SHOULD provide a mechanism for resolving merging conflicts ⌈

Description:	AUTOSAR authoring tools SHOULD provide a mechanism for resolving merging conflicts. This mechanism could be e.g. implemented by an interactive user interface which allows for choosing from conflicting elements or by using further meta-data such as the timestamp (e.g. the latest version of an element shall be used), etc.
Rationale:	Allow for integration of models which have been modified by different parties.
Use Case:	When integrating models which have been modified and extended by different parties merge conflicts are likely to occur.
Dependencies:	–
Supporting Material:	–

⌋

5.2.2.8 Handling merge conflicts: access control approach

The probability of merge conflicts can be reduced by a well defined work-flow. E.g. in a top down development process [UC_IOAT_00005] an OEM could decompose a system down to a given granularity. The refinement of the decomposition could be done by different suppliers. If each supplier modifies a disjunctive part of the model no merge conflicts will occur when merging the results into the OEMs model of the full system. In order to support this strategy the supplier needs to know which parts of a model are allowed to be changed and which parts are not allowed to be changed. These access rights SHOULD be exchanged together with meta-data for supporting data exchange. See requirement on the meta-data for data exchange [TR_IOAT_00036], [TR_IOAT_00038]. If access rights are defined an AUTOSAR authoring tool SHOULD prohibit the user from modifying model elements that are marked as read-only.

5.2.2.9 Authoring tool SHOULD prohibit the user from modifying model elements that are marked read-only

[TR_IOAT_00040] Authoring tool SHOULD prohibit the user from modifying model elements that are marked read-only ⌈

Description:	An AUTOSAR authoring tool SHOULD only allow the user to modify, create or delete model elements which he is allowed to change. The information if a model element is allowed to be modified should be represented to the user.
Rationale:	Prohibit a user from modifying model elements he is not allowed to change.
Use Case:	OEM wants to send information to tier-1 supplier. OEM wants to lock certain model elements so that the tier-1 is not allowed to change them.
Dependencies:	meta-data for data exchange could define the access-rights on model elements, see [TR_IOAT_00038]. The access-policies can support the merging of models, see [TR_IOAT_00042].
Supporting Material:	–

⌋

5.2.2.10 Example on merging models

Figure 5.12, Figure 5.13 and Figure 5.14 describe two example models and the result after merging them. The underlying meta-model for those examples is described in Figure 5.11. Splitable aggregations are marked by the stereotype ≪`atpSplitable`≫.

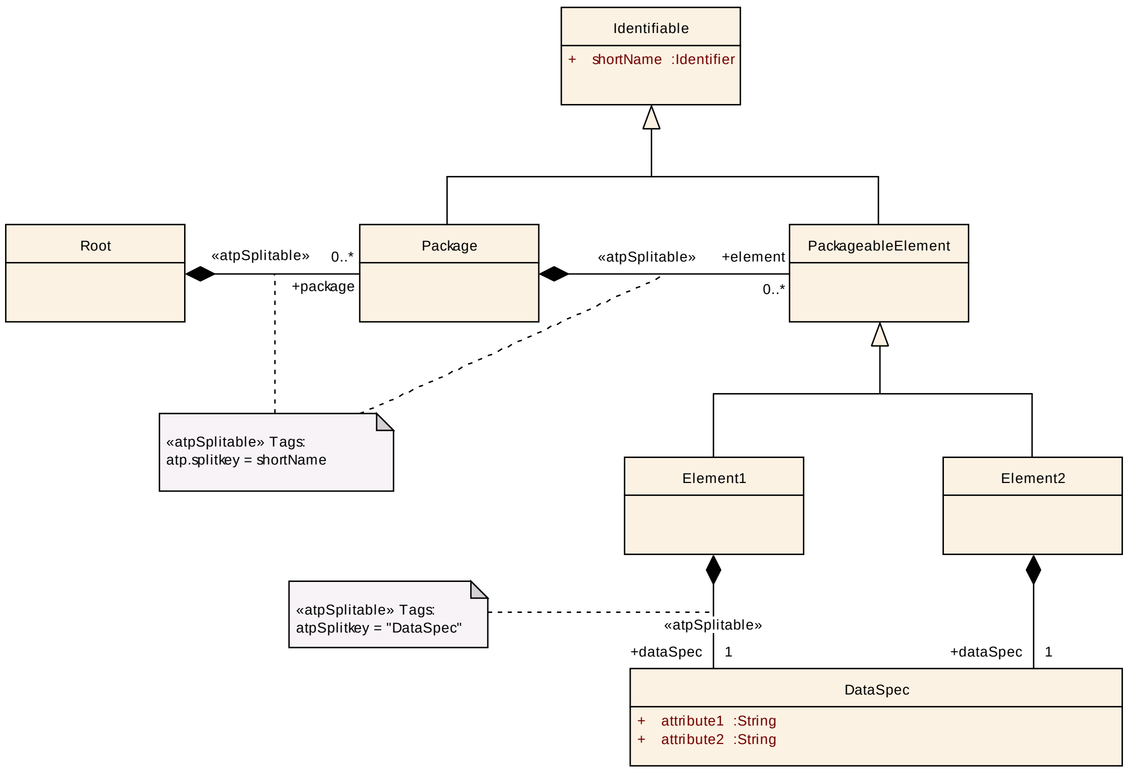

Figure 5.11: Meta-model of the models described in this section

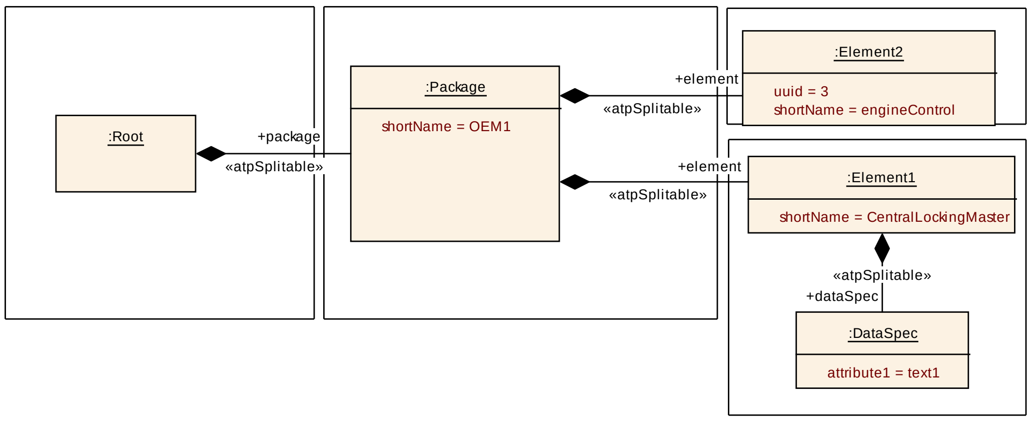

Figure 5.12: First example model

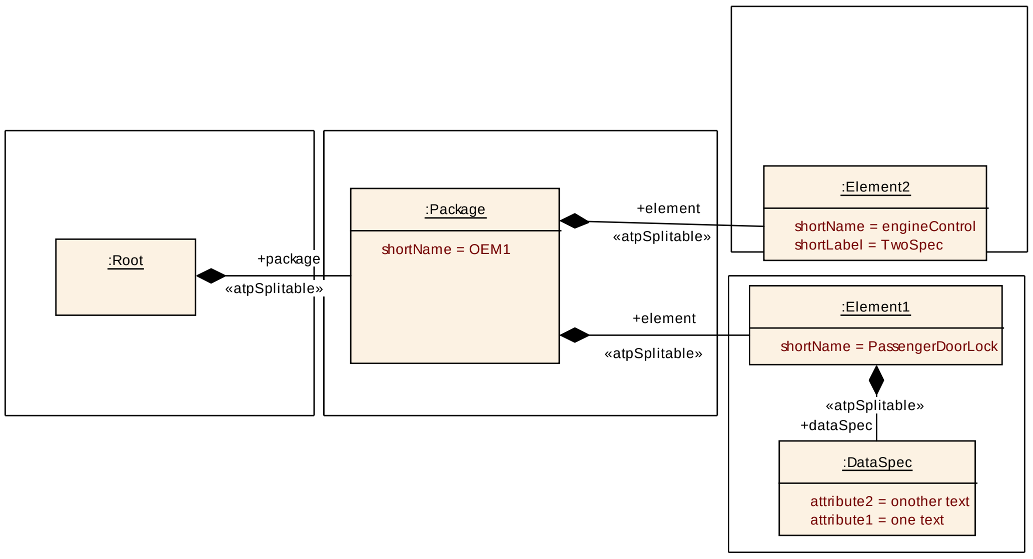

Figure 5.13: Second example model

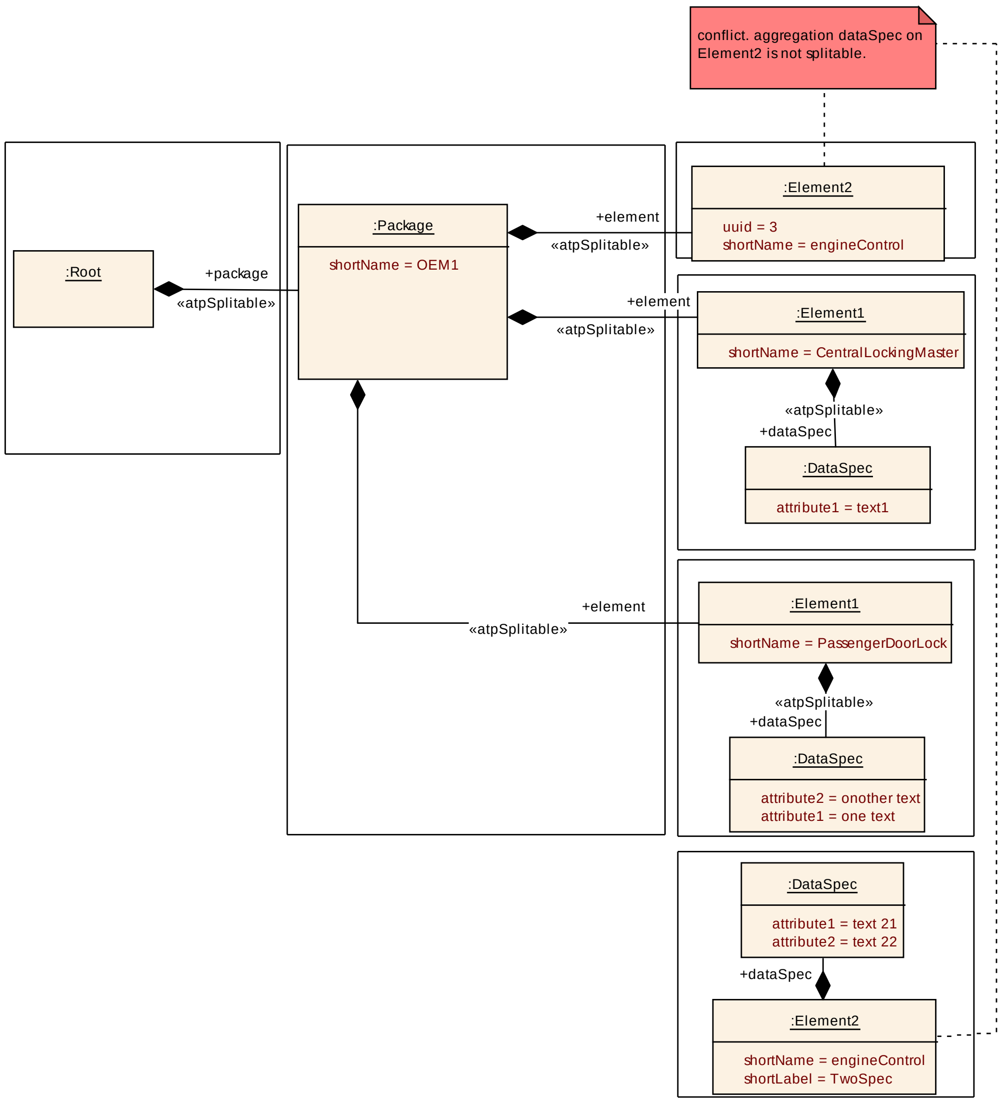

Figure 5.14: Result of merging the two example models

5.3 Shipment of AUTOSAR models and related artifacts

As described in the sections above, the minimum required data exchange mechanism of AUTOSAR authoring tools is the interpretation and creation of AUTOSAR XML descriptions which can be split up over several files.

The separation over several files imposes the risk that a file gets lost during the data exchange or a file was accidentally transmitted incompletely. Listing all files that belong to a shipment as meta-data for data exchange would allow the receiver to check if all files have been received.

The meta-data for data exchange should contain information that supports the exchange of AUTOSAR models and related artifacts:

- List of physical artifacts (e.g. files) that belong to the shipment in order to detect missing or superfluous artifacts. The physical artifacts are represented according to `EngineeringObject`.
- Checksum for each file in order to allow for detection of modification.
- Access rights on model elements which allow for transmitting information which model elements are allowed to be changed by the receiver.
- Explicit annotation of deleted, moved and added model elements in order to support merging.

5.3.1 AUTOSAR tool SHALL be able to interpret and create ASAM Container Catalog file for meta-data exchange

[TR_IOAT_00036] AUTOSAR tool SHALL be able to interpret and create ASAM Container Catalog file for meta-data exchange ⌈

Description:	An AUTOSAR tool SHALL be able to interpret and create ASAM Container Catalog file that contains meta-data about the relevant artfacts: • SHALL be able to load the partial models from the artifacts denoted in the catalog • SHALL be able to resolve `EngineeringObject` via the meta-data in the catalog to find the physical file names of the artifacts • SHOULD be able to interpret / maintain the `UPD`-Attribute in `<ABLOCK>` • SHOULD be able to only load a subset of the partial models depending on particular use cases
Rationale:	Support the exchange of additional information about exchanged XML-descriptions and their usage. e.g. access policies, checksum, list of files that contain the XML description. The Catalog also allows to find the relevant artifacts independent from the position in the file system.
Use Case:	[UC_IOAT_00008]

Dependencies:	More requirements on the meta-data for data exchange are described in [TR_IOAT_00037], [TR_IOAT_00038], [TR_IOAT_00039]. Per shipment all meta-data for data exchange must be stored in a single file. Meta-data for data exchange is not splitable. Otherwise meta-data for meta-data would be required.
Supporting Material:	ASAM container catalog [17]

⌋*(UC_IOAT_00008)*

Table 5.2 illustrates a (practicable) example for a set of `category`s for `EngineeringObject`s / `<ABLOCK>` for AUTOSAR artifacts containing partial models.

Example 5.4 illustrates a directory structure of AUTOSAR XML descriptions. The correlating ASAM CC is shown in example 5.5. Note that for the first entry (SysDescr "FillerCap") the representation of additional meta-data is illustrated.

Example 5.4

```
C:\TEMP\FILLERCAP
+-- FillerCap_cc.xml    <!-- the catalog file -->
+-- RBCentralElements_Standard.arxml
+-- FillerCap_SWCompo.arxml
+-- FillerCap_SysDescr.arxml
|
+-- FillerCapCompo
|   +-- FillerCapCompo_SWCD.arxml
|   \-- FillerCapCompo_Doc.arxml
|
+-- FCPCtrl
|   +-- FCPCtrl_SWCD.arxml
|   \-- FCPCtrl_Doc.arxml
|
+-- FCPIdctr
|   +-- FCPIdctr_SWCD.arxml
|   \-- FCPIdctr_Doc.arxml
|
+-- INTEG
|   +-- INTEG_SWCD.arxml
|   \-- INTEG_Doc.arxml
|
+-- IOIfc
|   +-- IOIfc_SWCD.arxml
|   \-- IOIfc_Doc.arxml
|
\-- SWTNrm
    +-- SWTNrm_SWCD.arxml
    \-- SWTNrm_Doc.arxml
```

Example 5.5

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

```
<!DOCTYPE CATALOG PUBLIC "-//ASAM//DTD␣CONTAINER␣CATALOG:V3.0.0:LAI:IAI:CC.
    DTD//EN" "catalog_V3_0_0.sl.dtd" >
<CATALOG>
  <SHORT-NAME>FillerCap</SHORT-NAME>
  <ADMIN-DATA>
    <LANGUAGE>en</LANGUAGE>
  </ADMIN-DATA>
  <ABLOCKS>
    <ABLOCK>
      <SHORT-NAME>FillerCap</SHORT-NAME>
      <CATEGORY>PRJ</CATEGORY>
      <AREFS>
        <AREF CATEGORY="SysDescr">FillerCap</AREF>
        <AREF CATEGORY="SWCompo">FillerCap</AREF>
        <AREF CATEGORY="Standard">RBCentralElements</AREF>
        <AREF CATEGORY="Doc">FillerCap</AREF>
        <AREF CATEGORY="SWCD">FillerCap</AREF>
        <AREF CATEGORY="Doc">SWTNrm</AREF>
        <AREF CATEGORY="SWCD">SWTNrm</AREF>
        <AREF CATEGORY="Doc">IOIfc</AREF>
        <AREF CATEGORY="SWCD">IOIfc</AREF>
        <AREF CATEGORY="Doc">INTEG</AREF>
        <AREF CATEGORY="SWCD">INTEG</AREF>
        <AREF CATEGORY="Doc">FCPIdctr</AREF>
        <AREF CATEGORY="SWCD">FCPIdctr</AREF>
        <AREF CATEGORY="Doc">FCPCtrl</AREF>
        <AREF CATEGORY="SWCD">FCPCtrl</AREF>
      </AREFS>
    </ABLOCK>
    <ABLOCK UPD="NEW">
      <SHORT-NAME>FillerCap</SHORT-NAME>
      <CATEGORY>SysDescr</CATEGORY>
      <ADMIN-DATA>
        <DOC-REVISIONS>
          <DOC-REVISION>
            <REVISION-LABEL>3.1.0</REVISION-LABEL>
            <ISSUED-BY>TheSender</ISSUED-BY>
          </DOC-REVISION>
        </DOC-REVISIONS>
        <SDGS>
          <SDG GID="AUTOSAR">
            <SD GID="MD5">B7296055E71F42807DF5FDB93EE0F50D</SD>
          </SDG>
        </SDGS>
      </ADMIN-DATA>
      <FILES>
        <FILE>FillerCap_SysDescr.arxml</FILE>
      </FILES>
    </ABLOCK>
    <ABLOCK>
      <SHORT-NAME>FillerCap</SHORT-NAME>
      <CATEGORY>SWCompo</CATEGORY>
      <FILES>
        <FILE>FillerCap_SWCompo.arxml</FILE>
      </FILES>
    </ABLOCK>
```

```
<ABLOCK>
  <SHORT-NAME>RBCentralElements</SHORT-NAME>
  <CATEGORY>Standard</CATEGORY>
  <FILES>
    <FILE>RBCentralElements_Standard.arxml</FILE>
  </FILES>
</ABLOCK>
<ABLOCK>
  <SHORT-NAME>FillerCapCompo</SHORT-NAME>
  <CATEGORY>SWCD</CATEGORY>
  <FILES>
    <FILE>FillerCapCompo/FillerCap_SWCD.arxml</FILE>
  </FILES>
</ABLOCK>
<ABLOCK>
  <SHORT-NAME>FillerCapCompo</SHORT-NAME>
  <CATEGORY>Doc</CATEGORY>
  <FILES>
    <FILE>FillerCapCompo/FillerCap_Doc.arxml</FILE>
  </FILES>
</ABLOCK>
<ABLOCK>
  <SHORT-NAME>SWTNrm</SHORT-NAME>
  <CATEGORY>SWCD</CATEGORY>
  <FILES>
    <FILE>SWNrm/SWTNrm_SWCD.arxml</FILE>
  </FILES>
</ABLOCK>
<ABLOCK>
  <SHORT-NAME>SWTNrm</SHORT-NAME>
  <CATEGORY>Doc</CATEGORY>
  <FILES>
    <FILE>SWNrm/SWTNrm_Doc.arxml</FILE>
  </FILES>
</ABLOCK>
<ABLOCK>
  <SHORT-NAME>IOIfc</SHORT-NAME>
  <CATEGORY>SWCD</CATEGORY>
  <FILES>
    <FILE>IOIfc/IOIfc_SWCD.arxml</FILE>
  </FILES>
</ABLOCK>
<ABLOCK>
  <SHORT-NAME>IOIfc</SHORT-NAME>
  <CATEGORY>Doc</CATEGORY>
  <FILES>
    <FILE>IOIfc/IOIfc_Doc.arxml</FILE>
  </FILES>
</ABLOCK>
<ABLOCK>
  <SHORT-NAME>INTEG</SHORT-NAME>
  <CATEGORY>SWCD</CATEGORY>
  <FILES>
    <FILE>INTEG/INTEG_SWCD.arxml</FILE>
  </FILES>
</ABLOCK>
```

```
    <ABLOCK>
      <SHORT-NAME>INTEG</SHORT-NAME>
      <CATEGORY>Doc</CATEGORY>
      <FILES>
        <FILE>INTEG/INTEG_Doc.arxml</FILE>
      </FILES>
    </ABLOCK>
    <ABLOCK>
      <SHORT-NAME>FCPIdctr</SHORT-NAME>
      <CATEGORY>SWCD</CATEGORY>
      <FILES>
        <FILE>FCPIdctr/FCPIdctr_SWCD.arxml</FILE>
      </FILES>
    </ABLOCK>
    <ABLOCK>
      <SHORT-NAME>FCPIdctr</SHORT-NAME>
      <CATEGORY>Doc</CATEGORY>
      <FILES>
        <FILE>FCPIdctr/FCPIdctr_Doc.arxml</FILE>
      </FILES>
    </ABLOCK>
    <ABLOCK>
      <SHORT-NAME>FCPCtrl</SHORT-NAME>
      <CATEGORY>SWCD</CATEGORY>
      <FILES>
        <FILE>FCPCtrl/FCPCtrl_SWCD.arxml</FILE>
      </FILES>
    </ABLOCK>
    <ABLOCK>
      <SHORT-NAME>FCPCtrl</SHORT-NAME>
      <CATEGORY>Doc</CATEGORY>
      <FILES>
        <FILE>FCPCtrl/FCPCtrl_Doc.arxml</FILE>
      </FILES>
    </ABLOCK>
  </ABLOCKS>
</CATALOG>
```

5.4 Naming convention of AUTOSAR XML description files

According to [TR_IOAT_00062] AUTOSAR XML descriptions SHALL use the file extension "`.arxml`"' (short for AUTOSAR XML).

[TR_IOAT_00069] Name pattern for AUTOSAR XML description files ⌈ AUTOSAR XML descriptions should use the following pattern for the file names:

`{module}_[{SubUseCaseAbbr}_]{UseCaseAbbr}.arxml`

- `{module}` describes the subject of the AUTOSAR XML description, for example one of
  - name of a module according to [TPS_GST_00017]

- the name of a component
- the name of the ECU
- even a fixed name like "SYS"

- {SubUseCaseAbbr} (optional) defines the intended sub-usage of the contained information.
- {UseCaseAbbr} defines the intended usage of the contained information. AUTOSAR has not formally defined the use cases for AUTOSAR XML description filenames. However, some examples are given by Table 5.2

The maximum length of the filename is restricted to 255 characters. The following restrictions apply:

```
name ::= [a-zA-Z][a-zA-Z0-9_]*
SubUseCaseAbbr ::= [a-zA-Z][a-zA-Z0-9]*
UseCaseAbbr ::= [a-zA-Z][a-zA-Z0-9]*
```

⌋

SubUseCaseAbbr	**UseCaseAbbr**	**Description**
	BSWMD	BSW Module Description
	Blueprint	Blueprints
	Doc	Documentation
	Example	Example objects
	EcuExtr	Ecu extract
STMD, VSMD	EcucParamDef	ECU Configuration Parameter Definition
	EcucValues	ECU Configuration Values
Prot	EcucValues	Protected ECU Configuration Values
Can, Lin, Flexray	Frames	Frames
	HWT	Hardware topology
	ISigs	Signals of the Interaction Layer
	Keyword	Keyword defintions
	PDUs	Protocol Data Unit
	SWCD	Component Type
	SWCompo	Ecu Software Composition
	Standard	Standardized objects
	SysDescr	System description
	SysExtr	System extract
	SysSigs	System Signals

Table 5.2: Use cases for AUTOSAR XML descriptions

[TR_IOAT_00070] Kind-based SubUseCase ⌈ Note that if one wants to use separate artifacts for the elements related to software component or module, it is recommended to use the kind as SubUseCaseAbbr. For example the CompuMethods for a component named foo, would be in an artifact named foo_CompuMethods_SWCT.arxml.
⌋

5.5 Specialized AUTOSAR tools

Within a tool-chain tools that consumes AUTOSAR models which have been created in earlier phases should be able to interpret and understand all information which has been defined before and is relevant for the processing step in question. This implies that an AUTOSAR tool chain may be built of tools which process only a subset of the AUTOSAR models. Therefore tools can be called AUTOSAR compliant, even if they are specialized to particular tasks and therefore only handle a special subset of the AUTOSAR meta-model.

5.5.1 Requirements for predictable tool interoperability

It should be predictable that two tools are able to exchange their models. In an early phase in the development process it should be possible to find out if two tools are able to exchange models which will be created in later phases. This could be a criterion for choosing tools that are used in an AUTOSAR development process.

The formal validation of this expression would require all AUTOSAR tools to formally describe the supported subset of the AUTOSAR meta-model. It needs to be checked if the structure and semantics of all meta-classes that are supported by the tool A is also supported by the tool B.

This approach would check if two tools can exchange information if all supported features are used. Usually not all features are always used and therefore the results of this formal compatibility check would become questionable.

A more pragmatic approach would be to compare features which describe the supported functionalities in a more abstract way. A starting point for these functionalities are the tasks defined in the "AUTOSAR methodology" document [2]. AUTOSAR should precisely define the required inputs and provided outputs of those tasks.

A more elaborate compatibility test can then be performed which can be based on a number of test-models that are created in early phases of the development of an AUTOSAR system.

In practice the information if two tools can exchange models is not the only criterion for choosing a special tool (consider criteria such as usability, know how with existing tools, etc). It is not very likely that e.g. a supplier switches to another tool because it is not completely compatible with the tool of an OEM. Instead he would approach the tool-vendor and ask for implementation of the missing features.

In order to get a set of AUTOSAR authoring tools which have comparable functionalities and are able to exchange their models, a set of compliance classes can help. (See Chapter 6.2 for more information on those compliance classes.)

5.5.1.1 Documentation of AUTOSAR tool SHOULD describe supported features

[TR_IOAT_00016] Documentation of AUTOSAR tool SHOULD describe supported features ⌈

Description:	The documentation of an AUTOSAR tool SHOULD describe the supported features. The description of the features SHOULD be based on the tasks defined in the "AUTOSAR methodology" document [2]
Rationale:	When choosing specialized Authoring tools that do not support the full set information described in the AUTOSAR meta-model for use in an AUTOSAR tool chain, it SHALL be predictable if AUTOSAR models can be exchanged between different tools.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

5.5.2 Requirements on the integration of specialized tools

The sections 5.1, 5.2 and 5.3 described some general concepts and requirements on tool interoperability. The following list describes some observations on handling of AUTOSAR models with special focus on tools that do not cover all information that is represented by the AUTOSAR meta-model:

- AUTOSAR models are exchanged via AUTOSAR XML descriptions which can be split up over several partial models. While interpreting an AUTOSAR XML description an AUTOSAR tool must merge the contents of the different partial models into the internal data-structure of the tool (see section 5.1.3).
- While merging AUTOSAR models conflicts can occur. Often user-interaction is required for the resolving of merge conflicts. A tool can usually only provide interaction mechanisms for model elements that are internally represented.
- While merging AUTOSAR models which were created by different parties (e.g.: integration of several independently developed models to an overall model) violations of semantic constraints can show up. Each AUTOSAR model could be valid in itself. However, violations can show up if all sub models are merged and checked together. These violations can only be solved by tools that are able to modify the violating information.
- A tool can very well perform its intended function even if some errors exist in the model. Only the information that is required for performing the intended function needs to be valid.
- **[TR_IOAT_00073] Utilize BuildActionManifest** ⌈ An AUTOSAR tool may be supported by `BuildActionManifest` (see [TPS_GST_00294]) to specify its

particular application. This can be supported by providing even blueprints of BuildActionManifest.⌋*(UC_IOAT_00008)*

5.6 Support for different versions of the meta-model

The implementation of the "Data Format Level", "Content level", "Semantic level" and all higher levels highly depend on the AUTOSAR meta-model. Changes in the meta-model can be classified in two different classes. The criteria for these classes are based on the effort that is usually required for adaptation of tools.

Minor changes Extensions in the meta-model that don't influence relations and constraints between existing classes in the meta-model are considered as minor changes. Existing models are still valid when used in a tool that conforms to a new version of the meta-model.

Major changes Changing the structure, semantics and/or the relations in the meta-model are considered as major changes. Existing models are no longer valid in newer versions of the tool. They need to be updated.
The following sections describe these two types of changes and their impact on the aforementioned levels in more details.

An AUTOSAR tool MAY be able to process AUTOSAR XML descriptions created according to different major versions of AUTOSAR. Therefore, it is feasible that content contributed by XML descriptions created according to different major versions can be freely combined inside the tool as long as the tool supports this approach. Nevertheless the default scenario for an AUTOSAR project is expected to use only one revision of XML schema.

Within a minor version, several revisions are created over time as a means to maintain the version. An AUTOSAR tool MAY but is not required to support all revisions of a specific version, e.g. it may support 4.0.1, 4.0.2, and 4.0.4. That is, 4.0.3 is left out in this example. In other words, tool vendors shall **not** be forced to upgrade all tools to support upcoming revisions because the changes implemented into a specific revision might be irrelevant for the particular tool.

Note that a tool that is e.g. unable to process an XML file created according to revision 4.0.3 MAY still load this file if it uses the XML Schema created for revision 4.0.4 as the basis for validation.

An AUTOSAR tool may be able to write AUTOSAR XML files according to a subset of the published revisions of a version. The tool is neither required to support **all** revisions for writing nor is it required to automatically write the file according to the latest revision.

An AUTOSAR tool MUST not change the AUTOSAR version of an XML description without permission by the user.

5.6.1 Minor changes in the meta-model

These changes don't influence the existing structure, interrelationships and constraints within the meta-model: Existing instances of an old meta-model remain valid with respect to a new meta-model. The following changes result in minor changes:

- adding new meta-classes,
- adding new (optional) aggregations,
- adding new (optional) references,
- adding new (optional) attributes,
- adding semantic constraints which do not effect existing meta-classes and relations and
- setting a class from abstract to concrete.

These minor changes in the meta-model SHOULD result in minor changes in the XML representation (existing XML descriptions remain valid with respect to a new AUTOSAR XML schema). AUTOSAR tools that are implemented according to a new version of the AUTOSAR meta-model (only minor changes have been performed) can interpret existing AUTOSAR XML descriptions. The following sections describe the effect on the different abstraction levels:

- "Physical level": changes on the meta-model don't have any affect on the physical level.
- "Data format level": The new AUTOSAR XML schema contains additional XML elements which represent the new content defined by the meta-model. Since AUTOSAR tools and the respective AUTOSAR XML schema shall allow for exchanging partially described AUTOSAR models[2], existing AUTOSAR XML descriptions remain valid with respect to the new AUTOSAR XML schema. In other words: It is only possible to exchange partially defined AUTOSAR XML descriptions if most of the content is considered to be optional in the AUTOSAR XML schema. Adding a new optional element doesn't violate the validity of existing AUTOSAR XML descriptions.
- "Content level": It is expected that new versions of the tool implementations are more powerful with respect to the coverage of meta-classes defined by the AUTOSAR meta-model. Therefore they will be able to interpret existing XML descriptions and create the internal data-representation.
- "Semantic level": The additional constraints are limited to the extensions. An existing AUTOSAR model is still valid with respect to the new constraints since these constraints have by definition of minor changes no impact on existing meta-classes.

[2]See requirement on AUTOSAR authoring tools [TR_IOAT_00035]

Please note: An old AUTOSAR tool can still fully interpret an AUTOSAR XML description if the new features have not been used in the description. Additionally, AUTOSAR tools are not required to support the full set of information defined in the meta-model (see section 5.5 for more details). They can ignore information they do not understand. Therefore an old tool could still interpret the supported features and ignore the additional features. Of course this old tool is usually not able to modify features that have been introduced in the extended meta-model.

An overview over the compatibility of AUTOSAR tools and AUTOSAR XML descriptions in case of **minor changes** is listed in Table 5.3.

	Old AUTOSAR tool	**New AUTOSAR tool**
Old AUTOSAR XML description	compatible	compatible
New AUTOSAR XML description (minor change)	Fully compatible only if new features are not used in the new XML description[3]. Otherwise an tool can still interpret the new XML description. However it is not able to modify information that was not available in the old AUTOSAR format.	compatible

Table 5.3: Compatibility matrix for minor changes

5.6.2 Major changes in the meta-model

Major changes are modifications on the meta-model which go beyond minor changes. These include:

- Adding constraints for existing meta-classes and relations
- Removing of meta-classes and relations
- Changes on the semantics
- Renaming of meta-classes or relations
- Changing the upper multiplicity of attributes, references or composite associations from 1 to a value bigger than one or vice versa (Those changes usually require changing the internal data structure of tools. Lists of elements need to be supported)
- Adding new specializations to meta-classes which did not already have specializations before

[3]Please note that the XML namespace might change in later versions of the AUTOSAR XML Schema. If the tool uses XML Schema validation it should be able to apply the old XML Schema to the new XML description even if the XML namespace has changed.

Existing AUTOSAR models can no longer be used without modifications in a tool that supports the new version of the meta-model. An AUTOSAR tool SHALL reject the import or provide means to automatically or manually update the model to the new version. Information on update should be given to the user.

Please note, that major changes in the meta-model only have an effect on a tool if the changed structures are supported by the tool. E.g. If a tool is specialized on the creation of software components and it doesn't support aspects described in the ECU Resource Template [6] then any major change in the meta-model concerning the ECU descriptions doesn't have any impact on that tool: From the point of view of this specialized tool, old AUTOSAR models remain compatible.

An overview over the compatibility of AUTOSAR tools and AUTOSAR XML descriptions in case of **major changes** is listed in Table 5.4.

	Old AUTOSAR tool	**New AUTOSAR tool**
Old AUTOSAR XML description	compatible	Compatible only if changes in the meta-model do not affect the content described in the old XML description. Otherwise an upgrade mechanism should be provided by the AUTOSAR tool.
New AUTOSAR XML description (major change)	Compatible only if changes in the meta-model do not affect the content described in the new XML description. Otherwise the tool SHOULD reject the XML description.	compatible

Table 5.4: Compatibility matrix for major changes

5.6.3 AUTOSAR tool SHALL properly handle Meta-Model versions

[TR_IOAT_00066] AUTOSAR tool SHALL properly handle Meta-Model versions ⌈

Description:	It should be predictable if tools (potentially with different underlying meta model versions) can exchange AUTOSAR models. • An AUTOSAR tool MAY but is not forced to support multiple versions of the AUTOSAR meta-model • An AUTOSAR authoring tools MUST not change an AUTOSAR xml description to another version of the AUTOSAR meta-model without user interaction.
Rationale:	The AUTOSAR meta model and the derived AUTOSAR data exchange format will change over time.
Use Case:	[UC_IOAT_00002]
Dependencies:	XML schema versions see [TR_IOAT_00012] and [TR_IOAT_00062]
Supporting Material:	–

⌋*(UC_IOAT_00002)*

5.6.4 Authoring tool SHOULD support upgrading AUTOSAR models

[TR_IOAT_00005] Authoring tool SHOULD support upgrading AUTOSAR models
⌈

Description:	AUTOSAR authoring tools SHOULD support upgrading AUTOSAR models from at least the last major version of the meta-model. It is not required that an authoring tool supports the upgrade of arbitrary AUTOSAR models. Only the content that is internally supported by the tool SHOULD be upgradeable. If the tool doesn't support automatic or manual upgrade of models then it SHALL reject the import of old XML description. It is not required that a tool creates AUTOSAR XML descriptions that are valid with respect to older AUTOSAR XML schema.
Rationale:	Reuse of existing AUTOSAR models.
Use Case:	[UC_IOAT_00002]
Dependencies:	–
Supporting Material:	–

⌋*(UC_IOAT_00002)*

5.7 Support for versioning of AUTOSAR models

5.7.1 Granularity of AUTOSAR models

The minimum granularity of an AUTOSAR model SHALL be defined in the meta-model. If aggregations in the meta-model are marked as ≪atpSplitable≫, then the aggregated elements may be stored in different models. Please note that each individual XML file SHALL be valid with respect to the AUTOSAR XML schema.

5.7.2 Annotation of AUTOSAR model elements by version information

The AUTOSAR meta-model supports meta-data for storing version information and authorship for each element that is derived from the meta-class Identifiable.

5.8 Standardized error handling

5.8.1 AUTOSAR tools SHALL perform a standardized error handling

[TR_IOAT_00065] AUTOSAR tools SHALL perform a standardized error handling
⌈

Description:	In order to be able to exchange information about errors in models it is required to have a common vocabulary for model errors. This allows for discussing about those errors while using different tools.
Rationale:	As a rationale for this proposal, please consider a scenario where different project partners carry out an AUTOSAR software project by means of different tools for e.g. structural design.
Use Case:	[UC_IOAT_00005], [UC_IOAT_00006], [UC_IOAT_00008]
Dependencies:	–
Supporting Material:	for more details see information below.

⌋*(RS_IOAT_00002, UC_IOAT_00005, UC_IOAT_00006, UC_IOAT_00008)*

Let, for example, developers at different organizations using different AUTOSAR tools work with an AUTOSAR model that contains semantic inconsistencies. Now let the error messages according to the inconsistencies be reported by the particular tools. How can the partners be sure that they talk about the same issue if each of the tools reports a different error without a hint to a standardized error case?

When working with an AUTOSAR authoring tool the validity of an AUTOSAR model can be checked at a wide variety of user interactions. E.g. the model could be checked whenever:

- AUTOSAR XML descriptions are interpreted or created,
- the user inserts new data,
- the user inserts some specific data or
- the user explicitly triggers a validation of the model.

According to [TR_IOAT_0003], an AUTOSAR tool SHALL support validity checks. Further details are not specified by AUTOSAR and are left over to the implementation of the tool.

All error messages SHALL be significant and meaningful enough that ECU developers (not only meta-model specialists) can solve the problem.

5.8.2 Error codes on semantic level

Authoring tools shall check the validity of AUTOSAR models against semantic constraints (see requirement [TR_IOAT_0003]). If semantic constraints are violated, an error-code needs to be created.

The semantic constraints are defined in the AUTOSAR template specifications according to the following pattern (see chapter 4.2.4):

[constr_<ConstraintId>] <ConstraintHeadline> ⌈ <Constraint description> ⌋

For each detected semantic constraint violation the following information shall be reported:

- **ConstraintId**: The Id mentioned in the template specification which SHALL be displayed in case the constraint is violated. Note that this ConstraintId shall be reported in the error message. The tool may introduce its own tool specific error ids which shall not be confused with any AUTOSAR ConstraintId.
- **ConstraintHeadline**: the formal definition of the constraint using the object constraint language.
- **ConstraintDescription**: The human readable description of the constraint.
- **Severity**: The severity fatal, critical, uncritical defines the behavior of the tool in case the constraint is violated[4].
- **path** Name and path of the element in the AUTOSAR XML description.

Note that an AUTOSAR Tool might evaluate additional constraints which are not (yet) expressed in the AUTOSAR template specifications. In this case it shall **not** define a ConstraintId outside of AUTOSAR. Otherwise such tool specific ConstraintId would potentially be in conflict with future AUTOSAR specifications.

5.8.3 Guidelines for standardized error reporting

5.8.3.1 Interactive authoring tool SHOULD guide the user to the locations of errors

[TR_IOAT_00049] Interactive authoring tool SHOULD guide the user to the locations of errors ⌈

Description:	An interactive authoring tool SHOULD guide the user to the locations of errors. If possible the error messages SHOULD contain a hint that describes how the error can be fixed.
Rationale:	Support user while fixing errors in an AUTOSAR model
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[4]As the severity is considered from the perspective of the tool, it is **not** specified with the constraints in the AUTOSAR templates

5.8.3.2 Authoring tool SHOULD support exchanging information about errors

[TR_IOAT_00050] Authoring tool SHOULD support exchanging information about errors ⌈

Description:	An AUTOSAR authoring tool SHOULD support exchanging information about errors.
Rationale:	Some tools (e.g.: highly specialized batch tools) might indicate an error but might only provide limited means for fixing the error. A standardized representation of information of the errors including the location in the model that can be read into another tool can help identifying and fixing the problems.
Use Case:	–
Dependencies:	[TR_IOAT_00055] meta-data for data exchange SHOULD contain information about errors in the model.
Supporting Material:	–

⌋

5.8.3.3 AUTOSAR tool MAY use well structured error messages

[TR_IOAT_00068] Authoring tool SHOULD support exchanging information about errors ⌈

Description:	AUTOSAR tool MAY use well structured error messages.
Rationale:	Provide as much information as suitable for particular error message. Support the recognition of relevant information in error message.
Use Case:	–
Dependencies:	this is an extension of [TR_IOAT_00049]
Supporting Material:	–

⌋

An AUTOSAR tool may structure is messages along its own purpose using specific fields. This may be represented even using XML and may be standardized in upcoming AUTOSAR versions.

The following list is a collection of proposed information items in particular applicable to log files used for exchanging information about errors.

ErrorCode A symbolic name for the message text

StandardErrorCode The reference to the AUTOSAR error code

ConstraintCode Reference to the semantic constraint mentioned in the AUTOSAR template specification.

Signature Signature of the message for duplicate checks

Timestamp A time stamp for the message

ShortName A unique identification which allows to refer to particular error messages. This can also be used to establish references between error messages, e.g. for Screening and also to trace back to root cause

Desc The human readable message text.

Component Such information item may help the user to locate the problem in the model

BaseUrl An url for a base directory which can be used as basis for file references in a log file. This is typically the root direactory of a project structure.

ColumNumber The column of the error position

LineNumber The line number of the error position

LongName The title of the error message

ObjectCategory The `category` of for example the involved `ApplicationPrimitiveDataType` (e.g.`VALUE`)

PrimaryErrorReference Reference to the root cause if applicable

ScopeEntryReference Reference to a scoping message if applicable

Object The shortName based reference to the AUTOSAR element which caused the error

ToolName The name of the tool which reported the error

ToolVersion The version of the tools which reported the error

IncidentUrl The Url which refers to the artifact in which the error occurs

Value The actual found value which caused the problem

5.9 Requirements on meta-data for data exchange

The availability of meta-data for data exchange supports the interoperability of AUTOSAR tools. This meta-data is e.g. required for checking the completeness of a shipment or for giving additional information on what the receiver of an AUTOSAR model is allowed to do with the model. This chapter describes requirements on the meta-data for data exchange.

5.9.1 Meta-data for data exchange SHALL be based on existing standards and SHALL be defined by AUTOSAR

[TR_IOAT_00037] Meta-data for data exchange SHALL be based on existing standards and SHALL be defined by AUTOSAR ⌈

Description:	The meta-data for supporting data exchange SHALL be based on existing standards.
Rationale:	Use of existing standardized (existing) solutions for interpretation and creation of meta-data.
Use Case:	Reduce costs for implementation of tools/libraries that are capable of interpreting and creating of meta-data for data exchange.
Dependencies:	–
Supporting Material:	Examples of existing standards for describing meta-data e.g. for data exchange are: ASAM Container Catalog - ASAM CC

⌋

5.9.2 Description of access rights SHOULD allow for being mapped to data structures that are different from the AUTOSAR meta-model

[TR_IOAT_00038] Description of access rights SHOULD allow for being mapped to data structures that are different from the AUTOSAR meta-model ⌈

Description:	The description of access rights SHOULD allow for being mapped to data structures that are different from the AUTOSAR meta-model.
Rationale:	The internal data structure of AUTOSAR tools will very likely be different to the structure of the AUTOSAR meta-model. Tools can only represent information about the access rights to the user if the access rights defined on instances of the AUTOSAR meta-model can be mapped to instances of the tool internal data structure.
Use Case:	A tool might want to represent a system on a high level of abstraction. E.g. the tool only shows some connections between a `CanCluster` and an `EcuInstance`. The existence of `PhysicalChannel`s and `CommConnectorPort` might be hidden to the user. If the access rights are defined in a very complex manner, then it might not be possible to decide which impact these right have to model elements in the tool with a different internal data structure.
Dependencies:	–
Supporting Material:	–

⌋

5.9.3 Meta-data for data exchange SHALL NOT change the content of AUTOSAR models

[TR_IOAT_00039] Meta-data for data exchange SHALL NOT change the content of AUTOSAR models ⌈

Description:	The meta-data for data exchange SHALL be independent from the content of an AUTOSAR model. The content of an AUTOSAR model SHALL remain identical if it is exchanged together with or without the meta-data.
Rationale:	Allow for integration of tools that do not support meta-data for data exchange. Avoiding the creation of new versions of the AUTOSAR model for each data exchange.
Use Case:	[UC_IOAT_00008]
Dependencies:	–
Supporting Material:	–

⌋

5.9.4 Meta-data for data exchange SHOULD contain information about errors in the model

[TR_IOAT_00055] Meta-data for data exchange SHOULD contain information about errors in the model ⌈

Description:	Meta-data for data exchange SHOULD contain information about errors in the model in a standardized format. This format SHOULD contain the standardized error code, the informal error message and the location of the detected error in the model.
Rationale:	Exchange information about errors contained in a model.
Use Case:	Within an AUTOSAR development process several kinds of tools might be used. E.g.: Interactive tools, batch tools, tools that do not support all elements and constraints in the meta-model. If error-log messages that are created by a specialized batch-tool are created in a standardized format, then this information can interpreted by other tools for fixing the errors.
Dependencies:	Error codes
Supporting Material:	–

⌋

5.9.5 Meta-data for data exchange SHOULD contain information about deleted, changed and moved elements

[TR_IOAT_00056] Meta-data for data exchange SHOULD contain information about deleted, changed and moved elements ⌈

Description:	Meta-data for data exchange SHOULD contain information that supports merging models. It SHOULD contain information about added, changed and moved elements.
Rationale:	Support automated merge without user-interaction.

Use Case:	Two models have been created out of a common model and contain redundant information. E.g.: both models contain an element with uuid=3. In one model that element has been removed explicitly. While merging the two models a tool needs to know if an element was removed explicitly or if one model was incomplete.
Dependencies:	–
Supporting Material:	–

⌋

6 Compliance

An AUTOSAR tool may be called "AUTOSAR compliant" if it implements the mandatory requirements on AUTOSAR authoring tools defined in this document. For better readability the requirements of this document are summarized in the following section.

6.1 Summary of requirements on AUTOSAR tools

The following table lists all requirements on AUTOSAR tools which are required for tool interoperability. Note that the mandatory requirements are indicated by the word SHALL (instead of SHOULD).

Each requirement is assigned to a number of abstraction levels (marked by "x"). This allows for e.g. easily identifying all requirements that are relevant on the content level and all higher level. For example, a plugin for an AUTOSAR authoring tool could directly access the internal data structure of the authoring tool: Plugin and authoring tool would exchange information on content level. For this communication requirements on lower levels are not relevant.

Requirement on AUTOSAR tools		**Abstraction Level**				
Mandatory Requirements	See Chapter	Physical	Data Format	Content	Semantic	Presentation
[TR_IOAT_00010] AUTOSAR tool SHALL support sets of files	5.1.1.1	x				
[TR_IOAT_00012] AUTOSAR tool SHALL support AUTOSAR XML descriptions	5.1.2.1		x			
[TR_IOAT_00033] Authoring tool SHALL be able to import and export supported model elements as AUTOSAR XML descriptions	5.1.2.2			x		
[TR_IOAT_00062] Authoring tool SHALL support well defined serialization	5.1.2.3		x			
[TR_IOAT_00007] Authoring tool SHALL NOT change model contents without the intention of the user	5.1.3.1			x		
[TR_IOAT_00035] Authoring tool SHALL support exchange of partial information	5.1.3.2			x		
[TR_IOAT_0003] Authoring tool SHALL support validity checks	5.1.4.1				x	
[TR_IOAT_00060] AUTOSAR tool SHALL support variants	5.1.4.2				x	
[TR_IOAT_00024] Authoring tool SHALL support unique identification of model elements	5.2.1.6			x		
[TR_IOAT_00061] Authoring tool SHALL be able to handle partial AUTOSAR models	5.2.2.1			x	x	
[TR_IOAT_00042] AUTOSAR tool SHALL support the merging of AUTOSAR models	5.2.2.2			x		
[TR_IOAT_00064] AUTOSAR tool SHALL resolve references	5.2.2.3			x		
[TR_IOAT_00063] AUTOSAR tools SHALL accept double defined `ARElement`s as long as their nonSplitables are the same	5.2.2.5			x		
[TR_IOAT_00036] AUTOSAR tool SHALL be able to interpret and create ASAM Container Catalog file for meta-data exchange	5.3.1	x		x		

[TR_IOAT_00066] AUTOSAR tool SHALL properly handle Meta-Model versions	5.6.3		x	x	x	x
[TR_IOAT_00065] AUTOSAR tools SHALL perform a standardized error handling	5.8.1		x	x	x	x
Optional Requirements						
[TR_IOAT_00048] Authoring tool SHOULD support AUTOSAR extension mechanism	5.1.3.3			x		
[TR_IOAT_00067] Authoring tool SHOULD maintain references	5.1.3.4			x		
[TR_IOAT_00043] Authoring tool SHOULD provide a mechanism for showing differences between AUTOSAR models	5.2.1.1			x		x
[TR_IOAT_00044] Authoring tool SHOULD provide a mechanism for resolving merging conflicts	5.2.2.7			x		x
[TR_IOAT_00040] Authoring tool SHOULD prohibit the user from modifying model elements that are marked read-only	5.2.2.9					x
[TR_IOAT_00016] Documentation of AUTOSAR tool SHOULD describe supported features	5.5.1.1					
[TR_IOAT_00005] Authoring tool SHOULD support upgrading AUTOSAR models	5.6.4		x	x	x	x
[TR_IOAT_00049] Interactive authoring tool SHOULD guide the user to the locations of errors	5.8.3.1					x
[TR_IOAT_00050] Authoring tool SHOULD support exchanging information about errors	5.8.3.2			x		
[TR_IOAT_00069] Naming convention for the AUTOSAR XML descriptions	5.4		x			

Table 6.1: Requirements on AUTOSAR tools

6.2 Notes on compliance

6.2.1 Compliance classes based on coverage of the meta-model

In order to allow for seamless tool interoperability, AUTOSAR tools should support a common set of features - the tools should implement a common coverage of the AUTOSAR meta-model.

Otherwise one tool would generate AUTOSAR models that cannot be interpreted by another tool. The definition of these common sets of features highly depends on the intended work-flow. The compliance classes should be based on the ability to perform tasks in the AUTOSAR Methodology [2].

6.2.2 Testing the compliance of an AUTOSAR authoring tool

The compliance of a given tool could be tested by defining a set of AUTOSAR XML descriptions which need to be processed by the AUTOSAR authoring tools. These compliance tests should be performed in early phases of an AUTOSAR system development by the stakeholders who need to exchange AUTOSAR models. This document does not define any models for testing the interoperability of AUTOSAR authoring tools.

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 ([19]).

At a high level, an artifact is represented as a single conceptual file.

AUTOSAR Tool This is a software tool which supports one or more tasks defined as AUTOSAR tasks in the methodology. Depending on the supported tasks, an AUTOSAR tool can act as an authoring tool, a converter tool, a processor tool or as a combination of those (see separate definitions).

AUTOSAR Authoring Tool An AUTOSAR Tool used to create and modify AUTOSAR XML Descriptions. Example: System Description Editor.

AUTOSAR Converter Tool An AUTOSAR Tool used to create AUTOSAR XML files by converting information from other AUTOSAR XML files. Example: ECU Flattener

AUTOSAR Definition This is the definition of parameters which can have values. One could say that the parameter values are Instances of the definitions. But in the meta model hierarchy of AUTOSAR, definitions are also instances of the meta model and therefore considered as a description. Examples for AUTOSAR definitions are: `EcucParameterDef`, `PostBuildVariantCriterion`, `SwSystemconst`.

AUTOSAR XML Description In AUTOSAR this means "filled Template". In fact an AUTOSAR XML description is the XML representation of an AUTOSAR model.

The AUTOSAR XML description can consist of several files. Each individual file represents an AUTOSAR partial model and shall validate successfully against the AUTOSAR XML schema.

AUTOSAR Meta-Model This is an UML2.0 model that defines the language for describing AUTOSAR systems. The AUTOSAR meta-model is an UML representation of the AUTOSAR templates. UML2.0 class diagrams are used to describe the attributes and their interrelationships. Stereotypes, UML tags and OCL expressions (object constraint language) are used for defining specific semantics and constraints.

AUTOSAR Model This is a representation of an AUTOSAR product. The AUTOSAR model represents aspects suitable to the intended use according to the AUTOSAR methodology.

Strictly speaking, this is an instance of the AUTOSAR meta-model. The information contained in the AUTOSAR model can be anything that is representable according to the AUTOSAR meta-model.

AUTOSAR Partial Model In AUTOSAR, the possible partitioning of models is marked in the meta-model by ≪`atpSplitable`≫. One partial model is represented in an AUTOSAR XML description by one file. The partial model does not need to fulfill all semantic constraints applicable to an AUTOSAR model.

AUTOSAR Processor Tool An AUTOSAR Tool used to create non-AUTOSAR files by processing information from AUTOSAR XML files. Example: RTE Generator

AUTOSAR Template The term "Template" is used in AUTOSAR to describe the format different kinds of descriptions. The term template comes from the idea, that AUTOSAR defines a kind of form which shall be filled out in order to describe a model. The filled form is then called the description.

In fact the AUTOSAR templates are now defined as a meta model.

AUTOSAR XML Schema This is a W3C XML schema that defines the language for exchanging AUTOSAR models. This Schema is derived from the AUTOSAR meta model. The AUTOSAR XML Schema defines the AUTOSAR data exchange format.

Blueprint This is a model from which other models can be derived by copy and refinement. Note that in contrast to meta model resp. types, this process is *not* an instantiation.

Instance Generally this is a particular exemplar of a model or of a type.

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.

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 History of Specification Items

B.1 History of Specification Items according to AUTOSAR R4.0.3

B.1.1 Added Specification Items in R4.0.3

Number	Heading
Use cases	
[UC_IOAT_00001]	Intergrate extracts from an AUTOSAR model of an OEM passed for further refinement and implementation to a supplier
[UC_IOAT_00002]	Dealing with changes of the AUTOSAR meta model over time
[UC_IOAT_00005]	Usage within the different steps of top-down functional development
[UC_IOAT_00006]	Support for direct exchange of AUTOSAR models in a tool-chain
[UC_IOAT_00008]	An AUTOSAR model and related artifacts are shipped from one party to another.
[UC_IOAT_00009]	Filter and merge AUTOSAR models
[UC_IOAT_00010]	Handling of identical double definitions
Requirements	
[TR_IOAT_00005]	Authoring tool SHOULD support upgrading AUTOSAR models
[TR_IOAT_00007]	Authoring tool SHALL NOT change model contents without the intention of the user
[TR_IOAT_00010]	AUTOSAR tool SHALL support sets of files
[TR_IOAT_00012]	AUTOSAR tool SHALL support AUTOSAR XML descriptions
[TR_IOAT_00016]	Documentation of AUTOSAR tool SHOULD describe supported features
[TR_IOAT_00024]	Authoring tool SHALL support unique identification of model elements
[TR_IOAT_00033]	Authoring tool SHALL be able to import and export supported model elements as AUTOSAR XML descriptions
[TR_IOAT_00035]	Authoring tool SHALL support exchange of partial information
[TR_IOAT_00036]	AUTOSAR tool SHALL be able to interpret and create ASAM Container Catalog file for meta-data exchange
[TR_IOAT_00037]	Meta-data for data exchange SHALL be based on existing standards and SHALL be defined by AUTOSAR
[TR_IOAT_00038]	Description of access rights SHOULD allow for being mapped to data structures that are different from the AUTOSAR meta-model
[TR_IOAT_00039]	Meta-data for data exchange SHALL NOT change the content of AUTOSAR models
[TR_IOAT_00040]	Authoring tool SHOULD prohibit the user from modifying model elements that are marked read-only
[TR_IOAT_00043]	Authoring tool SHOULD provide a mechanism for showing differences between AUTOSAR models
[TR_IOAT_00044]	Authoring tool SHOULD provide a mechanism for resolving merging conflicts
[TR_IOAT_00048]	Authoring tool SHALL support AUTOSAR extension mechanism
[TR_IOAT_00049]	Interactive authoring tool SHOULD guide the user to the locations of errors
[TR_IOAT_00050]	Authoring tool SHOULD support exchanging information about errors
[TR_IOAT_00055]	Meta-data for data exchange SHOULD contain information about errors in the model
[TR_IOAT_00056]	Meta-data for data exchange SHOULD contain information about deleted, changed and moved elements
[TR_IOAT_00060]	AUTOSAR tool SHALL support variants
[TR_IOAT_00061]	Authoring tool SHALL be able to handle partial AUTOSAR models
[TR_IOAT_00062]	Authoring tool SHALL support well defined serialization

[TR_IOAT_00063]	AUTOSAR tools SHALL accept double defined `ARElement`s as long as their nonSplitables are the same
[TR_IOAT_00064]	AUTOSAR authoring/integration tools SHALL resolve references
[TR_IOAT_00066]	AUTOSAR tool SHALL properly handle Meta-Model versions
[TR_IOAT_00067]	AUTOSAR Authoring tool SHOULD maintain references
[TR_IOAT_00068]	Authoring tool SHOULD support exchanging information about errors
[TR_IOAT_00069]	Name pattern for AUTOSAR XML description files
[TR_IOAT_00070]	Kind-based SubUseCase

Table B.1: Added Specification Items in R4.0.3

B.2 History of Specification Items according to AUTOSAR R4.1.1

B.2.1 Added Specification Items in R4.1.1

Number	Heading
[TR_IOAT_00071]	Basic Concepts of Data Exchange
[TR_IOAT_00072]	Support for AUTOSAR XML Data Exchange
[TR_IOAT_00073]	Utilize BuildActionManifest
[TR_IOAT_00074]	AUTOSAR Authoring tool SHOULD follow specified access rights

Table B.2: Added Specification Items in R4.1.1

C Mentioned Class Tables

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

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,Packageable Element,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table C.1: ARElement

Class	**ARPackage**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	AUTOSAR package, allowing to create top level packages to structure the contained ARElements. ARPackages are open sets. This means that in a file based description system multiple files can be used to partially describe the contents of a package. This is an extended version of MSR's SW-SYSTEM.			
Base	ARObject,AtpBlueprint,AtpBlueprintable,Collectable Element,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
arPackage	ARPackage	*	aggr	This represents a sub package within an ARPackage, thus allowing for an unlimited package hierarchy. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variation Point.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
element	PackageableEle ment	*	aggr	Elements that are part of this package **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variation Point.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=20

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
referenceBase	ReferenceBase	*	aggr	This denotes the reference bases for the package. This is the basis for all relative references within the package. The base needs to be selected according to the base attribute within the references. **Stereotypes:** atpSplitable **Tags:** atp.splitkey=shortLabel xml.sequenceOffset=10

Table C.2: ARPackage

Class	**ApplicationDataType (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	ApplicationDataType defines a data type from the application point of view. Especially it should be used whenever something "physical" is at stake. An ApplicationDataType represents a set of values as seen in the application model, such as measurement units. It does not consider implementation details such as bit-size, endianess, etc. It should be possible to model the application level aspects of a VFB system by using ApplicationDataTypes only.			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,AutosarDataType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table C.3: ApplicationDataType

Class	**ApplicationPrimitiveDataType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	A primitive data type defines a set of allowed values. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	ARElement,ARObject,ApplicationDataType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,AutosarDataType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table C.4: ApplicationPrimitiveDataType

Class	**AtomicSwComponentType (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	An atomic software component is atomic in the sense that it cannot be further decomposed and distributed across multiple ECUs.			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,Atp Type,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable,SwComponentType			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
internalBe havior	SwcInternalBeh avior	0..1	aggr	The SwcInternalBehaviors owned by an AtomicSwComponentType can be located in a different physical file. Therefore the aggregation is «atpSplitable». **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalBehavior, variation Point.shortLabel vh.latestBindingTime=preCompileTime
symbolPro ps	SymbolProps	0..1	aggr	This represents the SymbolProps for the AtomicSwComponentType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName

Table C.5: AtomicSwComponentType

Class	≪**atpVariation**≫ **CanCluster**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Can::CanTopology			
Note	CAN bus specific cluster attributes. **Tags:** atp.recommendedPackage=CommunicationClusters			
Base	ARObject,AbstractCanCluster,CollectableElement,CommunicationCluster,Fibex Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table C.6: CanCluster

Class	**Collection**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Element Collection			
Note	This meta-class specifies a collection of elements. A collection can be utilized to express additional aspects for a set of elements. Note that Collection is an ARElement. Therefore it is applicable e.g. for EvaluatedVariant, even if this is not obvious. Usually the category of a Collection is "SET". On the other hand, a Collection can also express an arbitrary relationship between elements. This is denoted by the category "RELATION" (see also [TPS_GST_00347]). In this case the collection represents an association from "sourceElement" to "targetElement" in the role "role". **Tags:** atp.recommendedPackage=Collections			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
autoCollect	AutoCollectEnum	0..1	attr	This attribute reflects how far the referenced objects are part of the collection. **Tags:** xml.sequenceOffset=20
collectedInstance	AtpFeature	*	iref	This instance ref supports the use case that a particular instance is part of the collection. **Tags:** xml.sequenceOffset=60
element	Identifiable	*	ref	This is an element in the collection. Note that Collection itself is collectable. Therefore collections can be nested. In case of category="RELATION" this represents the target end of the relation. **Tags:** xml.sequenceOffset=40
elementRole	Identifier	0..1	ref	This attribute allows to denote a particular role of the collection. Note that the applicable semantics shall be mutually agreed between the two parties. In particular it denotes the role of element in the context of sourceElement. **Tags:** xml.sequenceOffset=30
sourceElement	Identifiable	*	ref	Only if Category = "RELATION". This represents the source of a relation. **Tags:** xml.sequenceOffset=50
sourceInstance	AtpFeature	*	iref	Only if Category = "RELATION". This represents the source instance of a relation. **Tags:** xml.sequenceOffset=70

Table C.7: Collection

Class	**CommConnectorPort (abstract)**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreTopology			
Note	The Ecu communication relationship defines which signals, Pdus and frames are actually received and transmitted by this ECU. For each signal, Pdu or Frame that is transmitted or received and used by the Ecu an association between an ISignalPort, IPduPort or FramePort with the corresponding Triggering shall be created. An ISignalPort shall be created only if the corresponding signal is handled by COM (RTE or Signal Gateway). If a Pdu Gateway ECU only routes the Pdu without being interested in the content only a FramePort and an IPduPort needs to be created.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
communicationDirection	CommunicationDirectionType	1	attr	Communication Direction of the Connector Port (input or output Port).

Table C.8: CommConnectorPort

Class	**CompuMethod**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the ability to express the relationship between a physical value and the mathematical representation. Note that this is still independent of the technical implementation in data types. It only specifies the formula how the internal value corresponds to its physical pendant. **Tags:** atp.recommendedPackage=CompuMethods			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
compuInternalToPhys	Compu	0..1	aggr	This specifies the computation from internal values to physical values. **Tags:** xml.sequenceOffset=80
compuPhysToInternal	Compu	0..1	aggr	This represents the computation from physical values to the internal values. **Tags:** xml.sequenceOffset=90
displayFormat	DisplayFormatString	0..1	attr	This property specifies, how the physical value shall be displayed e.g. in documents or measurement and calibration tools. **Tags:** xml.sequenceOffset=20
unit	Unit	0..1	ref	This is the physical unit of the Physical values for which the CompuMethod applies. **Tags:** xml.sequenceOffset=30

Table C.9: CompuMethod

Class	**DataConstr**			
Package	M2::AUTOSARTemplates::CommonStructure::GlobalConstraints			
Note	This meta-class represents the ability to specify constraints on data. **Tags:** atp.recommendedPackage=DataConstrs			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,Collectable Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataConstr Rule	DataConstrRule	*	aggr	This is one particular rule within the data constraints. **Tags:** xml.roleElement=true; xml.roleWrapper Element=true; xml.sequenceOffset=30; xml.type Element=false; xml.typeWrapperElement=false

Table C.10: DataConstr

Class	**EcuInstance**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreTopology			
Note	ECUInstances are used to define the ECUs used in the topology. The type of the ECU is defined by a reference to an ECU specified with the ECU resource description. **Tags:** atp.recommendedPackage=EcuInstances			
Base	ARObject,CollectableElement,FibexElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
associated ComIPduG roup	ISignalIPduGro up	*	ref	With this reference it is possible to identify which ISignalIPduGroups are applicable for which CommunicationConnector/ ECU. Only top level ISignalIPduGroups shall be referenced by an EcuInstance. If an ISignalIPduGroup contains other ISignalIPduGroups than these contained ISignalIPduGroups shall not be referenced by the EcuInstance. Contained ISignalIPduGroups are associated to an EcuInstance via the top level ISignalIPduGroup.
associated PdurIPduG roup	PdurIPduGroup	*	ref	With this reference it is possible to identify which PduR IPdu Groups are applicable for which CommunicationConnector/ ECU.
canTpAddr ess	CanTpAddress	*	ref	Please note that this reference is deprecated and will be removed in future. A Tp Address can be assigned to an ECU without an existing TP Configuration. If TpNodes are described this reference shall not be used. **Tags:** atp.Status=obsolete; atp.StatusRevision Begin=4.1.3

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
comConfigurationGwTimeBase	TimeValue	0..1	attr	The period between successive calls to Com_MainFunctionRouteSignals of the AUTOSAR COM module in seconds.
comConfigurationRxTimeBase	TimeValue	0..1	attr	The period between successive calls to Com_MainFunctionRx of the AUTOSAR COM module in seconds.
comConfigurationTxTimeBase	TimeValue	0..1	attr	The period between successive calls to Com_MainFunctionTx of the AUTOSAR COM module in seconds.
comEnableMDTForCyclicTransmission	Boolean	0..1	attr	Enables for the Com module of this EcuInstance the minimum delay time monitoring for cyclic and repeated transmissions (TransmissionModeTiming has cyclicTiming assigned or eventControlledTiming with numberOfRepetitions > 0).
commController	CommunicationController	1..*	aggr	CommunicationControllers of the ECU.
connector	CommunicationConnector	*	aggr	All channels controlled by a single controller.
diagnosticAddress	Integer	0..1	attr	An ECU specific ID for responses of diagnostic routines.
partition	EcuPartition	*	aggr	Optional definition of Partitions within an Ecu.
sleepModeSupported	Boolean	1	attr	Specifies whether the ECU instance may be put to a "low power mode" • true: sleep mode is supported • false: sleep mode is not supported Note: This flag may only be set to "true" if the feature is supported by both hardware and basic software.
tpAddress	TpAddress	*	ref	Please note that this reference is deprecated and will be removed in future. A Tp Address can be assigned to an ECU without an existing TP Configuration. If TpNodes are described this reference shall not be used. **Tags:** atp.Status=obsolete; atp.StatusRevisionBegin=4.1.3
wakeUpOverBusSupported	Boolean	1	attr	Driver support for wakeup over Bus.

Table C.11: EcuInstance

Class	**EngineeringObject (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Engineering Object			
Note	This class specifies an engineering object. Usually such an object is represented by a file artifact. The properties of engineering object are such that the artifact can be found by querying an ASAM catalog file. The engineering object is uniquely identified by domain+category+shortLabel+revisionLabel.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
category	NameToken	1	attr	This denotes the role of the engineering object in the development cycle. Categories are such as • SWSRC for source code • SWOBJ for object code • SWHDR for a C-header file Further roles need to be defined via Methodology. **Tags:** xml.sequenceOffset=20
domain	NameToken	0..1	attr	This denotes the domain in which the engineering object is stored. This allows to indicate various segments in the repository keeping the engineering objects. The domain may segregate companies, as well as automotive domains. Details need to be defined by the Methodology. Attribute is optional to support a default domain. **Tags:** xml.sequenceOffset=40
revisionLabel	RevisionLabelString	*	attr	This is a revision label denoting a particular version of the engineering object. **Tags:** xml.sequenceOffset=30
shortLabel	NameToken	1	attr	This is the short name of the engineering object. Note that it is modeled as NameToken and not as Identifier since in ASAM-CC it is also a NameToken. **Tags:** xml.sequenceOffset=10

Table C.12: EngineeringObject

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

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

Table C.13: Identifiable

Class	**Implementation (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Description of an implementation a single software component or module.			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
buildAction Manifest	BuildActionMani fest	0..1	ref	A manifest specifying the intended build actions for the software delivered with this implementation. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime
codeDescri ptor	Code	1..*	aggr	Specifies the provided implementation code.
compiler	Compiler	*	aggr	Specifies the compiler for which this implementation has been released
generated Artifact	DependencyOn Artifact	*	aggr	Relates to an artifact that will be generated during the integration of this Implementation by an associated generator tool. Note that this is an optional information since it might not always be in the scope of a single module or component to provide this information. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
hwElement	HwElement	*	ref	The hardware elements (e.g. the processor) required for this implementation.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
linker	Linker	*	aggr	Specifies the linker for which this implementation has been released.
mcSupport	McSupportData	0..1	aggr	The measurement & calibration support data belonging to this implementation. The aggregtion is «atpSplitable» because in case of an already exisiting BSW Implementation model, this description will be added later in the process, namely at code generation time. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=mcSupport
programmingLanguage	ProgramminglanguageEnum	1	attr	Programming language the implementation was created in.
requiredArtifact	DependencyOnArtifact	*	aggr	Specifies that this Implementation depends on the existance of another artifact (e.g. a library). This aggregation of DependencyOnArtifact is subject to variability with the purpose to support variability in the implementations. Different algorithms in the implementation might cause different dependencies, e.g. the number of used libraries. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
requiredGeneratorTool	DependencyOnArtifact	*	aggr	Relates this Implementation to a generator tool in order to generate additional artifacts during integration. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
resourceConsumption	ResourceConsumption	1	aggr	All static and dynamic resources for each implementation are described within the ResourceConsumption class.
swVersion	RevisionLabelString	1	attr	Software version of this implementation. The numbering contains three levels (like major, minor, patch), its values are vendor specific.
swcBswMapping	SwcBswMapping	0..1	ref	This allows a mapping between an SWC and a BSW behavior to be attached to an implementation description (for AUTOSAR Service, ECU Abstraction and Complex Driver Components). It is up to the methodology to define whether this reference has to be set for the Swc- or BswImplementtion or for both.
usedCodeGenerator	String	0..1	attr	Optional: code generator used.
vendorId	PositiveInteger	1	attr	Vendor ID of this Implementation according to the AUTOSAR vendor list

Table C.14: Implementation

Class	**ImplementationDataType**			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. **Tags:** atp.recommendedPackage=ImplementationDataTypes			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,AutosarDataType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
subElement (ordered)	ImplementationDataTypeElement	*	aggr	Specifies an element of an arrray, struct, or union data type. The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a ImplementationDataType representing a structure. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the ImplementationDataType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table C.15: ImplementationDataType

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

Table C.16: Integer

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

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

Table C.17: Numerical

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

Table C.18: PackageableElement

Class	**PhysicalChannel (abstract)**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreTopology			
Note	A physical channel is the transmission medium that is used to send and receive information between communicating ECUs. Each CommunicationCluster has at least one physical channel. Bus systems like CAN and LIN only have exactly one PhysicalChannel. A FlexRay cluster may have more than one PhysicalChannels that may be used in parallel for redundant communication. An ECU is part of a cluster if it contains at least one controller that is connected to at least one channel of the cluster.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
commConnector	CommunicationConnector	1..*	ref	Reference to the ECUInstance via a CommunicationConnector to which the channel is connected. atpVariation: Variable assignment of Physical Channels to different CommunicationConnectors is expressed with this variation. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
frameTriggering	FrameTriggering	*	aggr	One frame triggering is defined for exactly one channel. Channels may have assigned an arbitrary number of frame triggerings. atpVariation: If signals/PDUs/frames are variable, the corresponding triggerings must be variable, too. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
iSignalTriggering	ISignalTriggering	*	aggr	One ISignalTriggering is defined for exactly one channel. Channels may have assigned an arbitrary number of ISignaltriggerings. atpVariation: If signals/PDUs/frames are variable, the corresponding triggerings must be variable, too. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
pduTriggering	PduTriggering	*	aggr	One PduTriggering is defined for exactly one channel. Channels may have assigned an arbitrary number of I-Pdu triggerings. atpVariation: If signals/PDUs/frames are variable, the corresponding triggerings must be variable, too. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild

Table C.19: PhysicalChannel

Class	**PhysicalDimension**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Units			
Note	This class represents a physical dimension. If the physical dimension of two units is identical, then a conversion between them is possible. The conversion between units is related to the definition of the physical dimension. Note that the equivalence of the exponents does not per se define the convertibility. For example Energy and Torque share the same exponents (Nm). Please note further the value of an exponent does not necessarily have to be an integer number. It is also possible that the value yields a rational number, e.g. to compute the square root of a given physical quantity. In this case the exponent value would be a rational number where the numerator value is 1 and the denominator value is 2. **Tags:** atp.recommendedPackage=PhysicalDimensions			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
currentExp	Numerical	0..1	attr	This attribute represents the exponent of the physical dimension "electric current". **Tags:** xml.sequenceOffset=50
lengthExp	Numerical	0..1	attr	The exponent of the physical dimension "length". **Tags:** xml.sequenceOffset=20
luminousIn tensityExp	Numerical	0..1	attr	The exponent of the physical dimension "luminous intensity". **Tags:** xml.sequenceOffset=80
massExp	Numerical	0..1	attr	The exponent of the physical dimension "mass". **Tags:** xml.sequenceOffset=30
molarAmo untExp	Numerical	0..1	attr	The exponent of the physical dimension "quantity of substance". **Tags:** xml.sequenceOffset=70
temperatur eExp	Numerical	0..1	attr	The exponent of the physical dimension "temperature". **Tags:** xml.sequenceOffset=60
timeExp	Numerical	0..1	attr	The exponent of the physical dimension "time". **Tags:** xml.sequenceOffset=40

Table C.20: PhysicalDimension

Class	**PortInterface (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Abstract base class for an interface that is either provided or required by a port of a software component.			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,Atp Type,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
isService	Boolean	1	attr	This flag is set if the PortInterface is to be used for communication between an • ApplicationSwComponentType or • ServiceProxySwComponentType or • SensorActuatorSwComponentType or • ComplexDeviceDriverSwComponentType or • EcuAbstractionSwComponentType and a ServiceSwComponentType (namely an AUTOSAR Service) located on the same ECU. Otherwise the flag is not set.
serviceKin d	ServiceProvider Enum	0..1	attr	This attribute provides further details about the nature of the applied service.

Table C.21: PortInterface

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,Multilanguage Referrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
clientServe rAnnotatio n	ClientServerAnn otation	*	aggr	Annotation of this PortPrototype with respect to client/server communication.
delegated PortAnnota tion	DelegatedPortA nnotation	0..1	aggr	Annotations on this delegated port.
ioHwAbstr actionServ erAnnotati on	IoHwAbstraction ServerAnnotatio n	*	aggr	Annotations on this IO Hardware Abstraction port.
modePortA nnotation	ModePortAnnot ation	*	aggr	Annotations on this mode port.
nvDataPort Annotation	NvDataPortAnn otation	*	aggr	Annotations on this non voilatile data port.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
parameterPortAnnotation	ParameterPortAnnotation	*	aggr	Annotations on this parameter port.
senderReceiverAnnotation	SenderReceiverAnnotation	*	aggr	Collection of annotations of this ports sender/receiver communication.
triggerPortAnnotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port.

Table C.22: PortPrototype

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

Table C.23: PortPrototypeBlueprint

Primitive	**Ref**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Primitive Types			
Note	This primitive denotes a name based reference. For detailed syntax see the xsd.pattern. • first slash (relative or absolute reference) [optional] • Identifier [required] • a sequence of slashes and Identifiers [optional] This primitive is used by the meta-model tools to create the references. **Tags:** xml.xsd.customType=REF; xml.xsd.pattern=/?[a-zA-Z][a-zA-Z0-9 _]{0,127}(/[a-zA-Z][a-zA-Z0-9_]{0,127})*; xml.xsd.type=string			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
base	Identifier	0..1	ref	This attribute reflects the base to be used for this reference. **Tags:** xml.attribute=true
index	PositiveInteger	0..1	attr	This attribute supports the use case to point on specific elements in an array. This is in particular required if arrays are used to implement particular data objects. **Tags:** xml.attribute=true

Table C.24: Ref

[constr_2552] Index attribute is only valid for arrays ⌈The index attribute in references is valid only if the reference target is an ApplicationArrayElement or if the reference target is an ImplementationDataTypeElement owned by an ImplementationDataType/ImplementationDataTypeElement of category ARRAY and has an attribute maxNumberOfElements/arraySize.⌋

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

Table C.25: Referrable

Class	**Sdg**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::SpecialData			
Note	Sdg (SpecialDataGroup) is a generic model which can be used to keep arbitrary information which is not explicitly modeled in the meta-model. Sdg can have various contents as defined by sdgContentsType. Special Data should only be used moderately since all elements should be defined in the meta-model. Thereby SDG should be considered as a temporary solution when no explicit model is available. If an sdgCaption is available, it is possible to establish a reference to the sdg structure.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
gid	NameToken	1	attr	This attributes specifies an identifier. Gid comes from the SGML/XML-Term "Generic Identifier" which is the element name in XML. The role of this attribute is the same as the name of an XML - element. **Tags:** xml.attribute=true
sdgCaption	SdgCaption	0..1	aggr	This aggregation allows to assign the properties of Identifiable to the sdg. By this, a shortName etc. can be assigned to the Sdg. **Tags:** xml.sequenceOffset=20
sdgCaptionRef	SdgCaption	0..1	ref	This association allows to reuse an already existing caption. **Tags:** xml.name=SDG-CAPTION-REF; xml.sequenceOffset=25
sdgContentsType	SdgContents	0..1	aggr	This is the content of the Sdg. **Tags:** xml.roleElement=false; xml.roleWrapper Element=false; xml.sequenceOffset=30; xml.type Element=false; xml.typeWrapperElement=false

Table C.26: Sdg

Class	**SenderReceiverInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A sender/receiver interface declares a number of data elements to be sent and received. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,Atp Type,CollectableElement,DataInterface,Identifiable,Multilanguage Referrable,PackageableElement,PortInterface,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataElement	VariableDataPrototype	1..*	aggr	The data elements of this SenderReceiverInterface.
invalidationPolicy	InvalidationPolicy	*	aggr	InvalidationPolicy for a particular dataElement

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

Table C.27: SenderReceiverInterface

Class	**SwBaseType**			
Package	M2::AUTOSARTemplates::CommonStructure::BaseTypes			
Note	This meta-class represents a base type used within ECU software. **Tags:** atp.recommendedPackage=BaseTypes			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,BaseType,Collectable Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table C.28: SwBaseType

Class	**Unit**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Units			
Note	This is a physical measurement unit. All units that might be defined should stem from SI units. In order to convert one unit into another factor and offset are defined. For the calculation from SI-unit to the defined unit the factor (factorSiToUnit) and the offset (offsetSiToUnit) are applied: unit = siUnit * factorSiToUnit + offsetSiToUnit For the calculation from a unit to SI-unit the reciprocal of the factor (factorSiToUnit) and the negation of the offset (offsetSiToUnit) are applied: siUnit = (unit - offsetSiToUnit) / factorSiToUnit **Tags:** atp.recommendedPackage=Units			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
displayName	SingleLanguage UnitNames	0..1	aggr	This specifies how the unit shall be displayed in documents or in user interfaces of tools.The displayName corresponds to the Unit.Display in an ASAM MCD-2MC file. **Tags:** xml.sequenceOffset=20
factorSiToUnit	Float	0..1	attr	This is the factor for the conversion from and to siUnits. **Tags:** xml.sequenceOffset=30
offsetSiToUnit	Float	0..1	attr	This is the offset for the conversion from and to siUnits. **Tags:** xml.sequenceOffset=40

Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
physicalDimension	PhysicalDimension	0..1	ref	This association represents the physical dimension to which the unit belongs to. Note that only values with units of the same physical dimensions might be converted. **Tags:** xml.sequenceOffset=50

Table C.29: Unit

Class	**ValueSpecification (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Base class for expressions leading to a value which can be used to initialize a data object.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
shortLabel	Identifier	0..1	ref	This can be used to identify particular value specifications for human readers, for example elements of a record type.

Table C.30: ValueSpecification

Class	**VariableDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	A VariableDataPrototype is used to contain values in an ECU application. This means that most likely a VariableDataPrototype allocates "static" memory on the ECU. In some cases optimization strategies might lead to a situation where the memory allocation can be avoided. In particular, the value of a VariableDataPrototype is likely to change as the ECU on which it is used executes.			
Base	ARObject,AtpFeature,AtpPrototype,AutosarDataPrototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype

Table C.31: VariableDataPrototype

Class	**VariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to express a "structural variation point". The container of the variation point is part of the selected variant if swSyscond evaluates to true and each postBuildVariantCriterion is fulfilled.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
desc	MultiLanguage OverviewParagr aph	0..1	aggr	This allows to describe shortly the purpose of the variation point. **Tags:** xml.sequenceOffset=20
blueprintC ondition	Documentation Block	0..1	aggr	This represents a description that documents how the variation point shall be resolved when deriving objects from the blueprint. Note that variationPoints are not allowed within a blueprintCondition. **Tags:** xml.sequenceOffset=28
formalBlue printCondit ion	BlueprintFormul a	0..1	aggr	This denotes a formal blueprintCondition. This shall be not in contradiction with blueprintCondition. It is recommanded only to use one of the two. **Tags:** xml.sequenceOffset=29
postBuildV ariantCond ition	PostBuildVarian tCondition	*	aggr	This is the set of post build variant conditions which all shall be fulfilled in order to (postbuild) bind the variation point. **Tags:** xml.sequenceOffset=40
sdg	Sdg	0..1	aggr	An optional special data group is attached to every variation point. These data can be used by external software systems to attach application specific data. For example, a variant management system might add an identifier, an URL or a specific classifier. **Tags:** xml.sequenceOffset=50
shortLabel	Identifier	0..1	ref	This provides a name to the particular variation point to support the RTE generator. It is necessary for supporting splitable aggregations and if binding time is later than codeGenerationTime, as well as some RTE conditions. It needs to be unique with in the enclosing Identifiables with the same ShortName. **Tags:** xml.sequenceOffset=10
swSyscon d	ConditionByFor mula	0..1	aggr	This condition acts as Binding Function for the VariationPoint. Note that the mulitplicity is 0..1 in order to support pure postBuild variants. **Tags:** xml.sequenceOffset=30

Table C.32: VariationPoint

Primitive	**VerbatimString**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Primitive Types			
Note	This primitive represents a string in which white-space needs to be preserved. **Tags:** xml.xsd.customType=VERBATIM-STRING; xml.xsd.type=string; xml.xsd.white Space=preserve			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
xmlSpace	XmlSpaceEnum	0..1	attr	This attribute is used to signal an intention that in that element, white space should be preserved by applications. It is defined according to xml:space as declared by W3C. **Tags:** atp.Status=shallBecomeMandatory xml.attribute=true; xml.attributeRef=true; xml.name=space; xml.nsPrefix=xml

Table C.33: VerbatimString