Document Title	Requirements on Interoperability of Autosar Tools
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	101
Document Classification	Auxiliary

Document Status	Final
Part of AUTOSAR Release	4.2.2

Document Change History

Release	Changed by	Description
4.2.2	AUTOSAR Release Management	• added use case section that was part of the TR_IOAT
4.2.1	AUTOSAR Release Management	• added requirement for naming conventions [RS_IOAT_00003] • minor editorial changes
4.1.1	AUTOSAR Administration	• improved requirements traceability • harmonized document structure
3.2.1	AUTOSAR Administration	• Legal disclaimer revised
3.1.1	AUTOSAR Administration	• Legal disclaimer revised
3.0.1	AUTOSAR Administration	• Document meta information extended • Small layout adaptations made
2.1.15	AUTOSAR Administration	• "'Advice for users"' revised • "'Revision information"' added
2.1	AUTOSAR Administration	• Legal disclaimer revised
2.0	AUTOSAR Administration	• Initial release

Disclaimer

Advice for users

AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

Bibliography

[1] Interoperability of AUTOSAR Tools
AUTOSAR_TR_InteroperabilityOfAutosarTools

[2] Meta Model
AUTOSAR_MMOD_MetaModel

[3] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[4] Main Requirements
AUTOSAR_RS_Main

[5] Requirements on AUTOSAR Features
AUTOSAR_RS_Features

[6] Methodology
AUTOSAR_TR_Methodology

[7] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate

[8] Container Catalog XML Model Specification
http://www.asam.net

1 Introduction

1.1 Scope of this document

This document collects the requirements on the `Interoperability of Autosar Tools` specification (IAOT) [1].

1.2 Terminology

1. The AUTOSAR metamodel[2] is a UML2.0 model that defines the language for describing AUTOSAR systems. The AUTOSAR metamodel is a graphical representation of a template. UML2.0 class diagrams are used to describe the attributes and their interrelationships. Stereotypes and OCL (object constraint language) are used for defining specific semantics and constraints.
2. An AUTOSAR model is an instance of the AUTOSAR metamodel. The information contained in the AUTOSAR model can be anything that is representable according to the AUTOSAR metamodel. The AUTOSAR model can be stored in many different ways: it might be a set of files in a file system, an XML stream, a database or memory used by some running software tools, etc.
3. The AUTOSAR XML Schema is a W3C XML schema that defines the language for exchanging AUTOSAR models. This Schema is derived from the AUTOSAR metamodel and defines the AUTOSAR data exchange format.
4. An AUTOSAR XML description describes the XML representation of an AUTOSAR model. The AUTOSAR XML description can consist of several fragments (e.g. files). Each individual fragment must validate successfully against the AUTOSAR XML Schema.
5. An AUTOSAR tool is a software tool which supports interpreting, processing and/or creating of AUTOSAR XML descriptions
6. An Metadata includes pertinent information about data, including information about the authorship, versioning, access-rights, timestamps etc.

1.3 Document Conventions

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([3]).

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([3]).

1.4 Guidelines

Existing specifications shall be referenced (in form of a single requirement). Differences to these specifications are specified as additional requirements. All Requirements shall have the following properties:

- Redundancy
  Requirements shall not be repeated within one requirement or in other requirements.
- Clearness
  All requirements shall allow one possibility of interpretation only. Used technical terms that are not in the glossary must be defined.
- Atomicity
  Each Requirement shall only contain one requirement. A Requirement is atomic if it cannot be split up in further requirements.
- Testability
  Requirements shall be testable by analysis, review or test.
- Traceability
  The source and status of a requirement shall be visible at all times.

2 Requirements Tracing

The following table references the requirements specified in [4], [5] and links to the fulfillments by this document.

Requirement	Description	Satisfied by
[RS_BRF_01028]	AUTOSAR shall provide naming conventions for symbols in its documentation	[RS_IOAT_00003]
[RS_Main_00300]	No description	[RS_IOAT_00001] [RS_IOAT_00002]

Table 2.1: RequirementsTracing

3 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 “UC_IOAT” (meaning Use Case - Interoperability of AUTOSAR Tools).

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 [6]).

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.

3.1 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). ([6])

  – “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]

3.2 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 [7]).

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 [8]) can indicate the changed artifacts.

A more fine-grain control can be performed using specific `Collection`s.

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

3.3 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.

⌋()

3.4 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`≫.

3.4.1 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 3.1).

- 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.

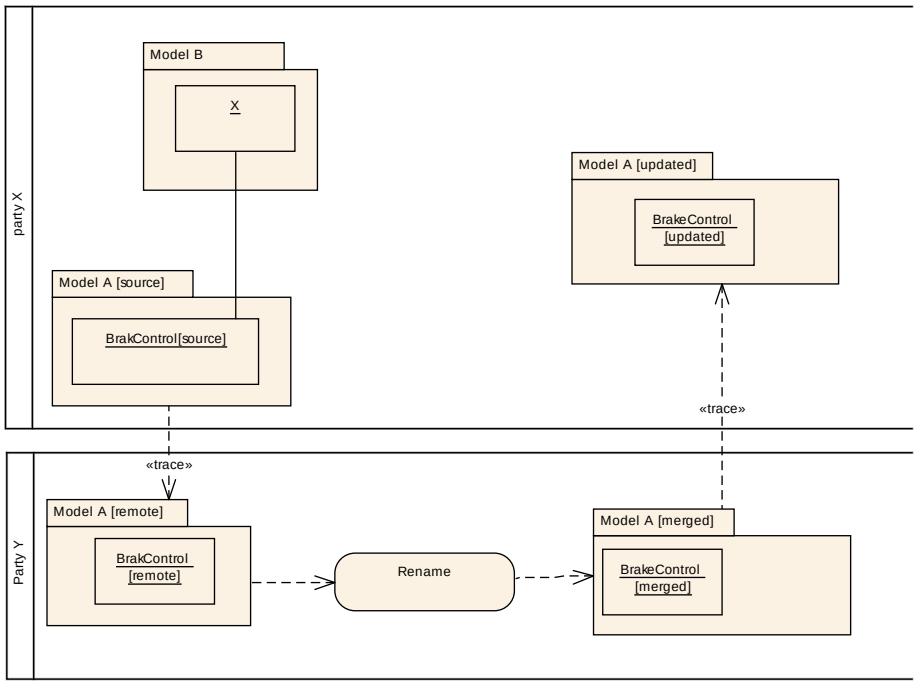

Figure 3.1: Concurrent modeling - renaming of elements

3.4.2 Updating of model elements

This scenario is similar to the renaming scenario; it differs only in the workflow (see Figure 3.2):

- 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

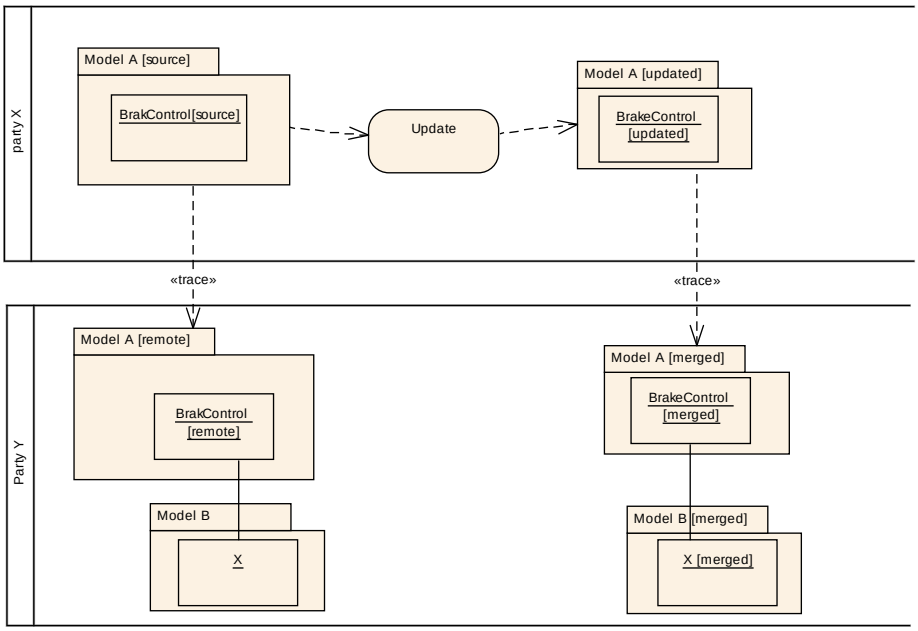

Figure 3.2: Concurrent modeling - updating of elements

3.4.3 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 3.4.1 and 3.4.2.

3.4.4 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.

3.5 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 au-

thoring 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.

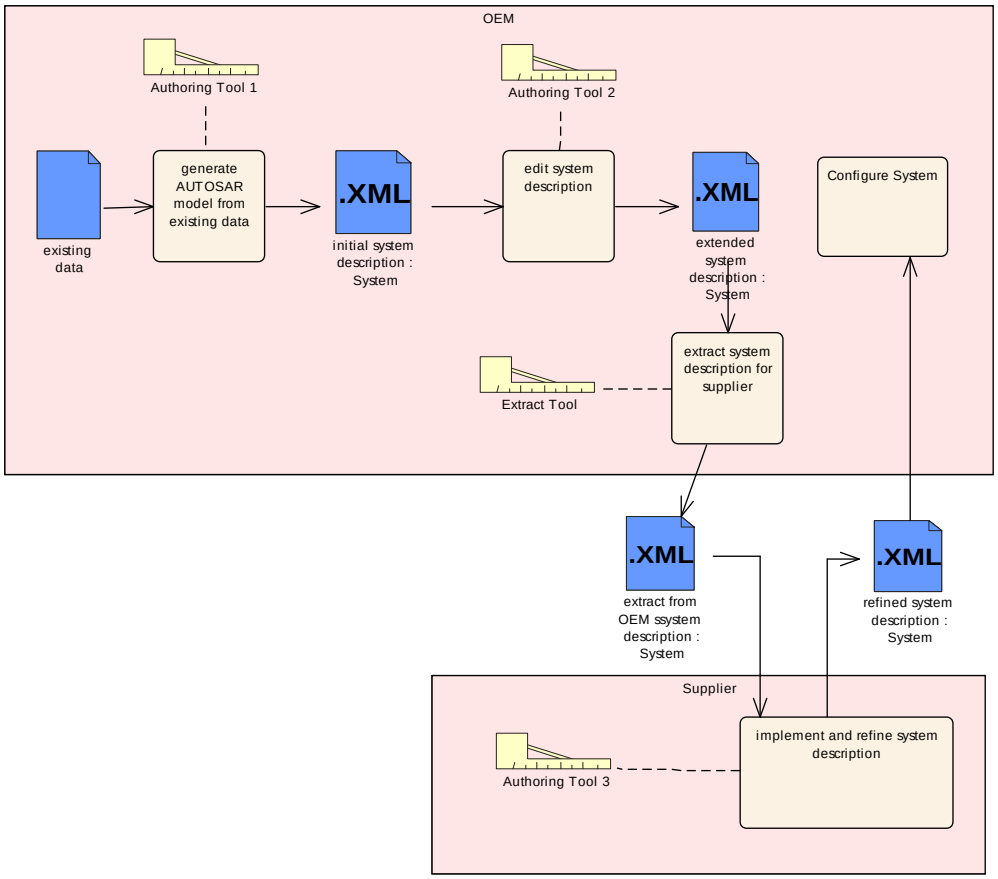

Figure 3.3: Tool Chain

3.6 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 `AtomicSwComponentTypes`. The implementation of the behavior has an impact on the `Implementation` description of `AtomicSwComponentTypes`. 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.

3.7 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. ⌋*()*

3.8 Handling of identical double definitions

[UC_IOAT_00010] Handling of identical double definitions ⌈ When working on one particular component, there are `ARElements` 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].

4 Requirements

This chapter provides a definition of the relevant requirements.

[RS_IOAT_00001] Support data exchange ⌈

Type:	valid
Description:	AUTOSAR SHALL define requirements on AUTOSAR tools AND requirements on the data exchange format which allow for seamless exchange of data between different AUTOSAR tools. The concept SHALL allow for exchanging of AUTOSAR models even if the AUTOSAR tools do not support all features defined in the AUTOSAR metamodel or methodology.
Rationale:	Within the AUTOSAR methodology AUTOSAR models will be exchanged between different parties. Each party could use different AUTOSAR tools which best fit to the step in the methodology. In order to facilitate seamless exchange of AUTOSAR models, a standardized AUTOSAR data exchange format is required. Additionally further requirements need to be defined on the AUTOSAR tools in order to keep AUTOSAR models consistent.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00300*)

[RS_IOAT_00002] Standardize the handling of errors in AUTOSAR models ⌈

Type:	valid
Description:	AUTOSAR SHALL provide a concept for a standardized mechanism for handling errors in AUTOSAR models. This concept SHALL not only be implemented by all AUTOSAR tools which interpret, modify or create AUTOSAR models.
Rationale:	Without a standard collection of possible errors, each tool would have its own sets, but the difference between these could cause relations created by one tool in a tool-chain be reported later as fatal errors by another tool.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00300*)

[RS_IOAT_00003] Provide naming conventions ⌈

Type:	valid
Description:	The TR_IAOT shall provide naming conventions. This especially includes requirement ids, module abbreviations, meta data and configuration symbols used in the document of a release and in AUTOSAR models
Rationale:	Avoid ambiguities and name clashes inside the specification and AUTOSAR models. Provide a consistent uniform presentation of meta data to the reader of the specification. Allow automatic processing of specification elements. Improve Interoperability between AUTOSAR tools.

Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_BRF_01028*)