Document Title	Specification of Abstract Platform
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	947

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

Document Change History

Date	Release	Changed by	Description
2021-11-25	R21-11	AUTOSAR Release Management	• CompositeInterface renamed to ApplicationInterface • support for attributes in an ApplicationInterface
2020-11-30	R20-11	AUTOSAR Release Management	• Migration of document to standard Foundation • Restructuring and further conceptual detailing • Addition of several Appendix examples
2019-11-28	R19-11	AUTOSAR Release Management	• Initial release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

References

[1] Glossary
AUTOSAR_TR_Glossary

[2] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[3] Virtual Functional Bus
AUTOSAR_EXP_VFB

[4] Meta Model
AUTOSAR_MMOD_MetaModel

[5] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate

[6] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[7] Methodology for Adaptive Platform
AUTOSAR_TR_AdaptiveMethodology

1 Introduction

1.1 Document Structure

This document contains the specification of the design of an AUTOSAR abstract platform (XP). Due to the specification being abstract of the AUTOSAR adaptive platform (AP) and AUTOSAR classic platform (CP), it is released as part of the AUTOSAR foundation (FO).

The document is structured in the following way:

Section 1 (this chapter) documents the terms, abbreviations, conventions; scope and limitations in the specification and requirement tracing.

Section 2 provides a description of the big picture, sets the background reasons and motivation for the specification and usage principles for intended stakeholders. Additionally, the general modeling approach and modeling decisions are described.

Section 3 dives into the design aspects of an *abstract platform*. The modeling is described along with constraints and specification items. The sub-sections detail the methodology, the modeling of the VFB elements and the data types.

Section 4 annotation and traceability of requirements.

Section A example ARXML listings and models.

1.2 Terms and Abbreviations

The main list of terms and abbreviations are defined in [1]. The following table contains the list of terms and abbreviations used in the scope of this document which are not already defined in [1] along with the spelled-out meaning of each of the abbreviations.

Term/Abbreviation	*Meaning*
ASD	Abstract Platform System Description
RSI	REST Services Interface
SYSML	Systems Modelling Language
VFB++	Abstract Platform VFB
VIWI	Volkswagen Infotainment Web Interface
XSC	Abstract Software Component

Table 1.1: Terms and Abbreviations used in the scope of this Document

1.3 Document Conventions

Technical terms are typeset in mono spaced font, e.g. `PortPrototype`. As a general rule, plural forms of technical terms are created by adding "s" to the singular form, e.g. `PortPrototype`s. By this means the document resembles terminology used in the AUTOSAR XML Schema.

This document contains constraints in textual form that are distinguished from the rest of the text by a unique numerical constraint ID, a headline, and the actual constraint text starting after the ⌈ character and terminated by the ⌋ character.

The purpose of these constraints is to literally constrain the interpretation of the AUTOSAR meta-model such that it is possible to detect violations of the standardized behavior implemented in an instance of the meta-model (i.e. on M1 level).

Makers of AUTOSAR tools are encouraged to add the numerical ID of a constraint that corresponds to an M1 modeling issue as part of the diagnostic message issued by the tool.

The attributes of the classes introduced in this document are listed in form of class tables. They have the form shown in the example of the top-level element AUTOSAR:

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

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

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

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

Let `AUTOSAR` be an example of a typical class table. The first rows in the table have the following meaning:

Class: The name of the class as defined in the UML model.

Package: The UML package the class is defined in. This is only listed to help locating the class in the overall meta model.

Note: The comment the modeler gave for the class (class note). Stereotypes and UML tags of the class are also denoted here.

Base Classes: If applicable, the list of direct base classes.

The headers in the table have the following meaning:

Attribute: The name of an attribute of the class. Note that AUTOSAR does not distinguish between class attributes and owned association ends.

Type: The type of an attribute of the class.

Mul.: The assigned multiplicity of the attribute, i.e. how many instances of the given data type are associated with the attribute.

Kind: Specifies, whether the attribute is aggregated in the class (aggr aggregation), an UML attribute in the class (attr primitive attribute), or just referenced by it (ref reference). Instance references are also indicated (iref instance reference) in this field.

Note: The comment the modeler gave for the class attribute (role note). Stereotypes and UML tags of the class are also denoted here.

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

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([2]).

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

1.4 Scope and Limitations

In the AUTOSAR timeline, the XP specification was added after the CP and AP. It is also independent of the existing CP/APs, and for that reason, it is released as part of the AUTOSAR Foundation.

The XP uses the terms: VFB and VFB++. The AUTOSAR VFB is conceptually described in [3]. While that document resides in the CP, the general principles in [3] chapter "Overall mechanisms and concepts" also apply to AP and XP. In particular the idea of a VFB level view applies in modeling terms to the set of those meta-model artifacts i.e. components, ports, interfaces, connectors used to describe the functional inter-ECU communications. This is independent of whether the platform under discussion is signal based, service based or abstract.

An XP VFB++ description is a purely software functional design description. It is independent of topology and deployment and thus does not describe these.

An XP description has its technical borders. The basis of an XP description shall be an AUTOSAR System description. This fits together with general methodology of

AUTOSAR to root the description of an AUTOSAR system in an own description, see 3.2.

The scope of the XP system description is, on VFB level, from the outlining of the SWC design down to the detailing of the definition of application level data types in the software interfaces, see 3.5.

See [TPS_SWCT_01229], [TPS_SWCT_01230] and [TPS_SWCT_01236] for details on application level data types.

2 Concept

2.1 Background

The existing AUTOSAR meta-model provides a means to comprehensively design and deploy applications on CP ECUs and AP Machines. Depending on the intended chosen platform for concrete deployment, the feature/function design model is (intentionally) tightly coupled to the choice of platform.

A system designer is drawn in advance into a concrete decision whether to design and deploy on AP or CP or indeed non-AUTOSAR platform. The design choices become therefore biased by the intended deployment platform.

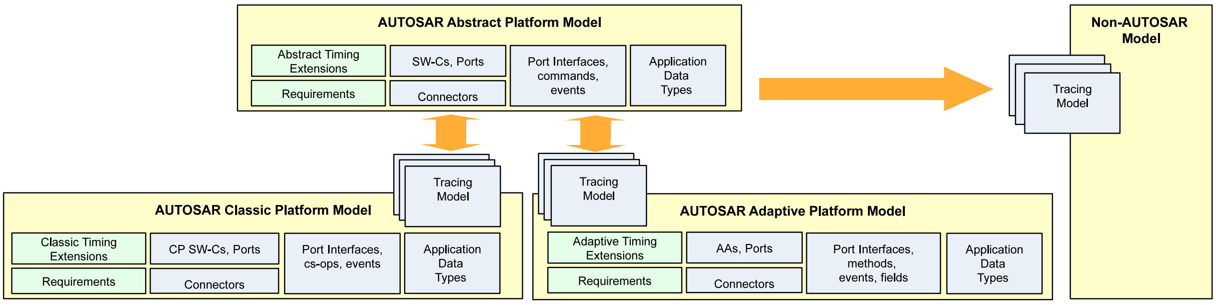
AUTOSAR Abstract Platform Model
Abstract Timing Extensions
Requirements
SW-Cs, Ports
Connectors
Port Interfaces, commands, events
Application Data Types
Non-AUTOSAR Model
Tracing Model
Tracing Model
Tracing Model
AUTOSAR Classic Platform Model
Classic Timing Extensions
Requirements
CP SW-Cs, Ports
Connectors
Port Interfaces, cs-ops, events
Application Data Types
AUTOSAR Adaptive Platform Model
Adaptive Timing Extensions
Requirements
AAs, Ports
Connectors
Port Interfaces, methods, events, fields
Application Data Types

Figure 2.1: Placement of an abstract platform

An system designer at an early **software** design stage may not necessarily care about for example what type of concrete component shall implement the function, or, which type of concrete interface provides the required data.

Rather the designer just wants to model the interaction between the functional software blocks and specify the basics: i.e. signal names, the directional flow of the data (providers/consumers) and the physical data types. Further refinement of the design will be done in a downstream stage, i.e. separation of concerns.

In methodology terms, this dovetails quite neatly to the whole design approach of AUTOSAR - whereby typically a staged approach to design is used. Foreseeably, this would be generally more suited to a *green-fields* or *blank-page* design methodology implemented in the OEM[1] - in contrast to other types of design methodology, e.g. where the supplier has very limited technical design decision.

[1] and progressing through various stages of refinement ending in a design finalization (by either in-house or external supplier)

2.2 Usage

2.2.1 General

The specification aims to provide a system description of a functional model. It further allows requirement annotation and general traceability of model elements including requirements and functional elements. The abstract description may provide a higher level view of a system, to help a system designers "step back" from early decisions about deployment, or indeed whether to defer that decision to a downstream design stage or to a supplier(s).

While the principal use-cases are founded for AP and CP, it is not (by design) intended to be exclusive to those platforms. Usage with *other* automotive or non-automotive domains should also be possible as shown in Figure 2.2.

Standardized Non-AUTOSAR systems have their own domain-specific models/IDLs and it is not within the scope of the XP to try to determine what these domain-specific models/IDLs are - or indeed, whether they should be accommodated in the scope of the XP. Rather, the intention is that through domain-specific tooling some form of model-to-model translation/derivation can be done from an XP description to a non-AUTOSAR model.

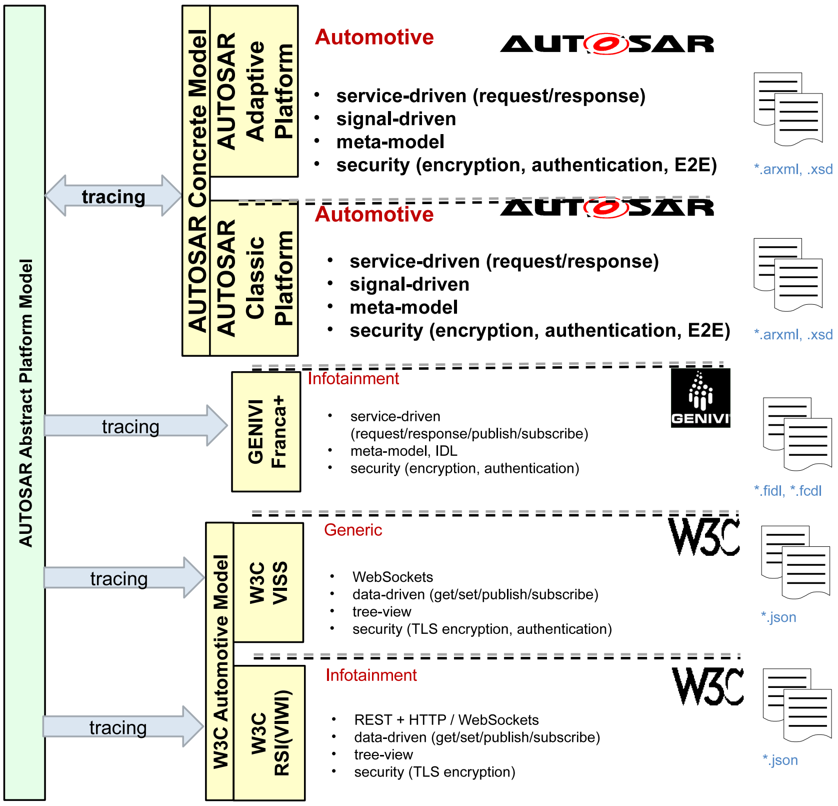
AUTOSAR Abstract Platform Model
tracing
tracing
tracing
tracing
AUTOSAR Concrete Model
AUTOSAR Adaptive Platform
AUTOSAR Classic Platform
GENIVI Franca+
W3C Automotive Model
W3C VISS
W3C RSI(VIWI)
Automotive
AUTOSAR
service-driven (request/response)
signal-driven
meta-model
security (encryption, authentication, E2E)
*.arxml, .xsd
Automotive
AUTOSAR
service-driven (request/response)
signal-driven
meta-model
security (encryption, authentication, E2E)
*.arxml, .xsd
Infotainment
GENIVI
service-driven (request/response/publish/subscribe)
meta-model, IDL
security (encryption, authentication)
*.fidl, *.fcdl
Generic
W3C
WebSockets
data-driven (get/set/publish/subscribe)
tree-view
security (TLS encryption, authentication)
*.json
Infotainment
W3C
REST + HTTP / WebSockets
data-driven (get/set/publish/subscribe)
tree-view
security (TLS encryption)
*.json

Figure 2.2: General relationship between abstract and concrete level standards

2.2.2 AUTOSAR Specific

There is not a *hard* modeling dependency between an XP and a AP/CP platform view in the sense that the concrete level depends on the abstract. The methodological approach does not forbid a system designer bypassing entirely an XP model and designing only in an AP/CP model to achieve the desired result[2].

Nevertheless, with the support of tooling and tracing, it should be similarly entirely possible to create an XP out of an AP/CP description.

An example scenario is shown in Figure 2.3. An abstract platform model with several levels of compositions of XSCs of different flavors, is derived to parallel AP/CP models. In this case a split of the XP model is shown, but in general, any number of permutations could be possible include a full derivation to either platform.

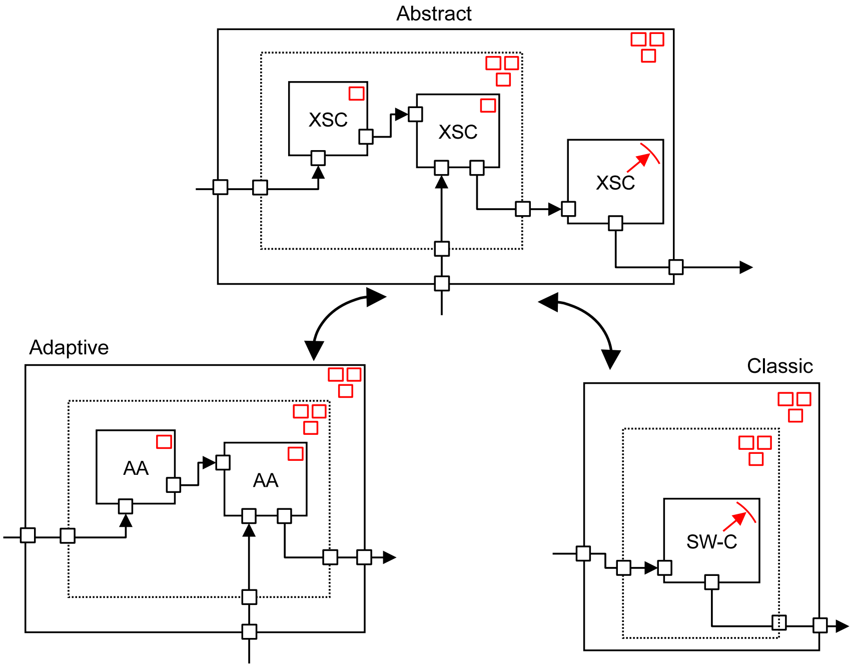

Figure 2.3: Relationship between AUTOSAR Abstract, Classic and Adaptive models

2.3 Modeling Approach

2.3.1 Meta-Model Choice

If the goal is to allow an abstract design, it could be argued that the chosen M2 model should also be abstract (of AUTOSAR). However, while the abstract design should be open to designers of *non-AUTOSAR platforms* to utilize, the primary focus is usage within the AUTOSAR domain i.e. AP/CP.

[2] backwards compatibility

For that reason, the argumentation of using the AUTOSAR Meta-Model [4] as the basis for the M2 level modeling approach is solidified. The XP is designed using the AUTOSAR meta-model, but should not restrict usage of abstract designs to AUTOSAR.

2.3.2 Bottom-up vs Top-down

Based on the assumption of the meta-model choice in 2.3.1, the next point is how to approach the creation of an abstract platform model. In very general terms, there are two possible approaches: bottom-up and top-down. Note: AUTOSAR supports tracing between models with dedicated meta-model artifacts in [5] chapter "Documentation Support::Documentation Block".

2.3.2.1 Top-down

If an XP model is created top-down, the VFB++ functional interactions are modeled using a *green fields* approach - this abstract model is then traced through to the creation of a new concrete model.

If the concrete model shall be an AUTOSAR model this involves deriving the VFB++ view to the VFB view in the respective AP/CP. If the concrete model is a non-AUTOSAR model it is in the domain of the non-AUTOSAR model to define this.

While this approach offers more freedom to design, there is a risk of specifying an XP which, in the end, is too distant from the needs of the existing platforms. The more likely approach therefore is to favor the bottom-up method.

2.3.2.2 Bottom-up

If an XP model is created by bottom-up, an existing concrete platform model is taken as the basis for the content. This in practice means that this form of XP description is immediately *more* valid than the former approach because it already has a basis in a concrete platform model. This approach would also allow for an automated creation of an XP description.

If the concrete model is an AP and CP model, the existing VFB view in the AP/CP platforms should be abstracted upstream to create the XP VFB++ model. If the concrete model is a non-AUTOSAR model it is in the domain of the non-AUTOSAR model to define this.

With this approach, the XP design is better guaranteed to fit well with the existing platforms.

2.3.3 Meta-class selection

Having decided on the general approach for the design of the XP, the next question is which approach to use regarding meta-class selection, i.e. re-use existing meta-classes or create new meta-classes.

While the AP and CP are based on different architecture principles, they mostly share the same modeling principles on VFB level and thus the VFB modeling. The approach is therefore to examine the VFB level model in both platforms as a primary basis and the non-AUTOSAR platforms as a secondary basis.

The existing AUTOSAR meta-model, especially the specification of the AUTOSAR Software-Component Template [6] already provides a good basis to comprehensively design a software component model. The principles therein may also be found in other more generic non-AUTOSAR component models.

It may be that any given identical meta-class may be used in any of the XP, AP or CP platforms. This approach is similar to that used when designing the AP meta-model, and similarly, it is necessary to either extend meta-classes with XP specifics and constrain them to the XP ([TPS_GST_00372]).

3 Abstract Platform

3.1 Methodology

3.1.1 Overview

An abstract platform system description provides the possibility to achieve a higher-level software view on the system. An architect can decide during design time which type of downstream AUTOSAR system description to use.

A level of architectural freedom through abstraction is attained by formally describing the functional interactions on a *component model* level, but without fixing details of any downstream implementation platform.

[TPS_APSD_01000]{DRAFT} **Principle of an abstract platform system description** ⌈An abstract platform system description allows a platform independent specification of the functional interactions of inter-connected software components.⌋*()*

[TPS_APSD_01001]{DRAFT} **VFB level modeling of an abstract platform** ⌈An abstract platform description uses those VFB level elements in the AUTOSAR meta-model as the basis for modeling.⌋*()*

[TPS_APSD_01002]{DRAFT} **Agnosticism of deployment aspects** ⌈An abstract platform is agnostic of deployment aspects.⌋*()*

AUTOSAR CP/AP models are still the basis for platform specific software design and should remain independent of an XP. To preserve the separation of concerns, it should be avoided that AUTOSAR CP/AP models use XP artifacts. The inverse case (usage of CP/AP in an XP) however, is allowed.

This is enforced by utilizing the tagging mechanism to place platform specific restrictions on those XP artifacts to exclude their visibility in CP/AP models. Refer to [TPS_-GST_00372] in [5] chapter "Usage of UML in AUTOSAR Templates::UML Tags" for an explanation of tagging.

[TPS_APSD_01035]{DRAFT} **Placement of an abstract platform model** ⌈An abstract platform model is wholly independent of concrete platform models.⌋*()*

[TPS_APSD_01003]{DRAFT} **Exclusion of abstract platform artifacts to an AUTOSAR concrete platform** ⌈The abstract platform uses the AUTOSAR mmt.RestrictToStandards tag to exclude abstract platform meta-model artifacts from other platforms.⌋*()*

As seen in 3.1, in a methodological view the XP System Description is decoupled completely from the equivalent System Descriptions in CP and AP.

Currently, there is no further detailed specification of e.g. roles, sub-tasks or deliverables (this may be added in a future release) - it is therefore advised to apply the same methodology steps from [7] since they are in most cases aligned.

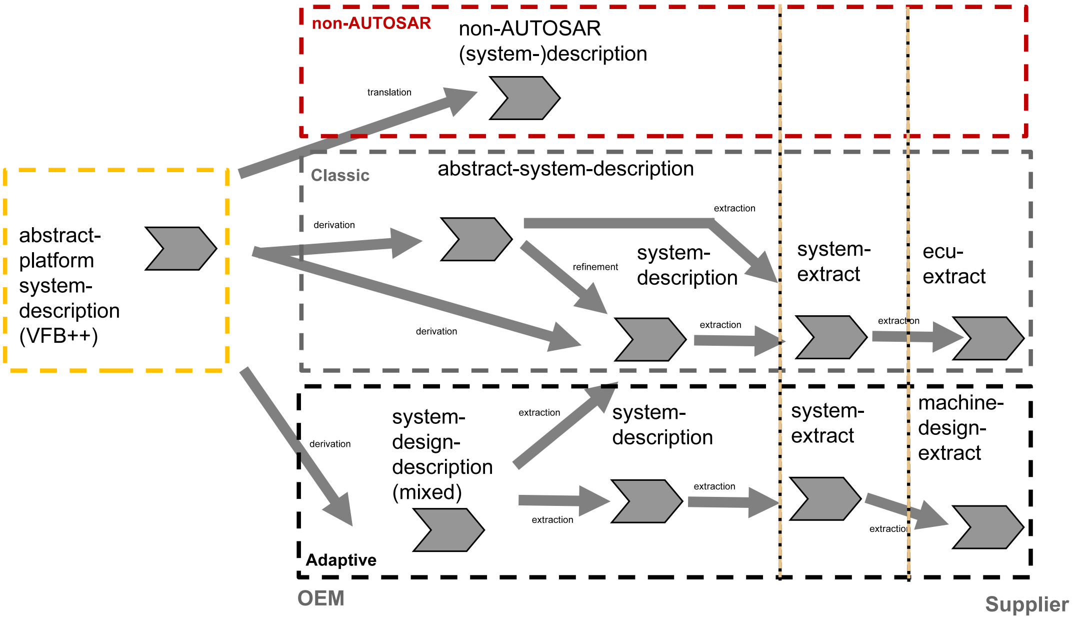

Figure 3.1: Abstract Platform System Description in Methodology

3.2 System Model

3.2.1 Overview

As per existing system descriptions in AP and CP, an XP needs its own system description to distinguish XP content from other types of system descriptions/extracts. The basis for all AUTOSAR system descriptions/extracts is the meta-class System and as with other AUTOSAR system descriptions, the category shall be used to identify the content.

[TPS_APSD_01004]{DRAFT} **System category for a system description with Abstract Platform content** ⌈The System element that contains design artifacts that are relevant for an Abstract Platform shall have the category:

- ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION.

⌋*()*

See A.1 for an example ARXML listing.

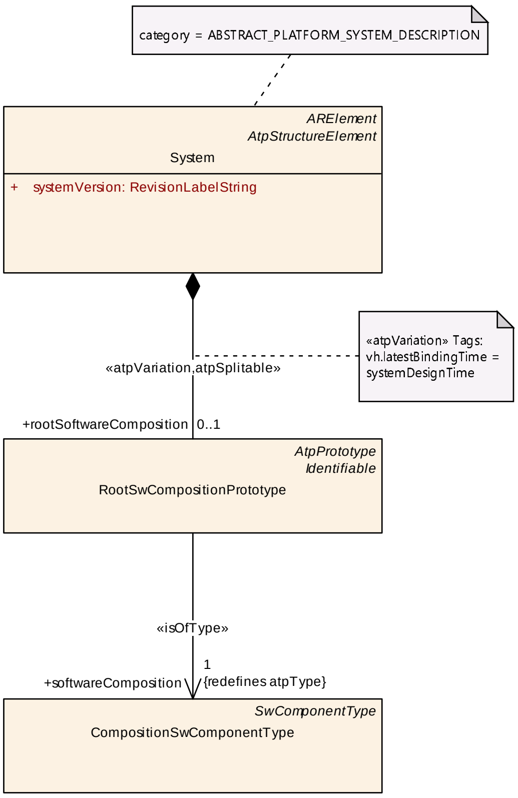

Figure 3.2: Modeling of an Abstract Platform System

3.2.2 Root Composition

As with other types of Systems in AP and CP, the RootSwCompositionPrototype in an XP references a CompositionSwComponentType as the root composition. With reference to [TPS_APSD_01019], this has the semantics of just a plain old composition.

Class	**System**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Note	The top level element of the Abstract Platform System Description. **Tags:**atp.recommendedPackage=Systems			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

mapping	SystemMapping	*	aggr	**Stereotypes:** atpSplitable; atpVariation
				Tags:
				atp.Splitkey=mapping.shortName, mapping.variationPoint.shortLabel
				vh.latestBindingTime=postBuild

▽

△

Class	**System**			
rootSoftware Composition	RootSwComposition Prototype	0..1	aggr	Aggregation of the root software composition, containing all software components in the System in a hierarchical structure. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=rootSoftwareComposition.shortName, rootSoftwareComposition.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime
systemVersion	RevisionLabelString	1	attr	Version number of the System Description.

Table 3.1: System

Class	**RootSwCompositionPrototype**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Note	The RootSwCompositionPrototype represents the top-level-composition of software components within a given System. This may for example be a more or less complete VFB++ description. Therefore the RootSwComposition will only occasionally contain all atomic software components that are used in a complete VFB System. The OEM is primarily interested in the required functionality and the interfaces defining the integration of the Software Component into the System. The internal structure of such a component contains often substantial intellectual property of a supplier. Therefore a top-level software composition will often contain empty compositions which represent subsystems. The contained SwComponentPrototypes are fully specified by their SwComponentTypes (including Port Prototypes, PortInterfaces, VariableDataPrototypes, etc.).			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
software Composition	CompositionSw ComponentType	1	tref	We assume that there is exactly one top-level composition that includes all Component instances of the system. **Stereotypes:** isOfType

Table 3.2: RootSwCompositionPrototype

3.3 Software Component Model

3.3.1 Overview

The XP software component model follows generally the aspects laid out in [6] chapter "Overview::Software Components...". The principles of reusability of SWCs and the type-prototype pattern are applicable in an XP, albeit in most cases, with a more restrictive view than the concrete platforms. An example of this more restrictive view is in the permitted types of software components shown later.

Another feature taken over from [6] is the ability to specify a hierarchy of SWCs of arbitrary complexity. In contrast to the CP software component model which specifies precise atomic SWC types with precise use cases in mind; the XP relaxes this kind of precision and targets a more generic typing of SWCs.

The XP SWC model allows a component design which does not force any intended downstream usage to the designer, but nevertheless allows a limited set of indicators [TPS_APSD_01005] to identify the intended usage of the component.

3.3.2 Component Compositions

This [TPS_APSD_01006] is no different than in AP and CP which handle encapsulation of SWCs the same. The modeling principles of compositions and encapsulation are suitably explained in [6] chapters "Composition::Overview" and "Composition::SwComponentPrototype" and do not need to be further detailed here.

[TPS_APSD_01006]{DRAFT} **Recursive component definition in an abstract platform** ⌈An abstract component design allows recursive depth-wise definition of components.⌋*()*

3.3.2.1 SwComponentPrototypes

The meta-class CompositionSwComponentType aggregates SwComponentPrototype in the role component which facilitates the modeling of an arbitrary nesting of components of SwComponentTypes. However, the XP only utilizes CompositionSwComponentTypes as the contained type.

[TPS_APSD_01019]{DRAFT} **Typing of SwComponentPrototypes used in a CompositionSwComponentType in an abstract platform** ⌈The SwComponentPrototype.type aggregated in a CompositionSwComponentType shall be CompositionSwComponentType in an abstract platform.⌋*()*

3.3.3 Component Types

In an XP, an designer should have the freedom to design a VFB++ hierarchical SWC model in a rather free-floating manner. The SWCs can be described by utilizing a subset of intended component types.

The approach is to allow a modeling of a generic type of component, CompositionSwComponentType as the generic component type. In other words, an abstract platform component is not bound to a specific use case. The CompositionSwComponentType inherits from the category attribute from Identifiable which means it can be assigned a subset of a categorys to specify the functional intent of the component.

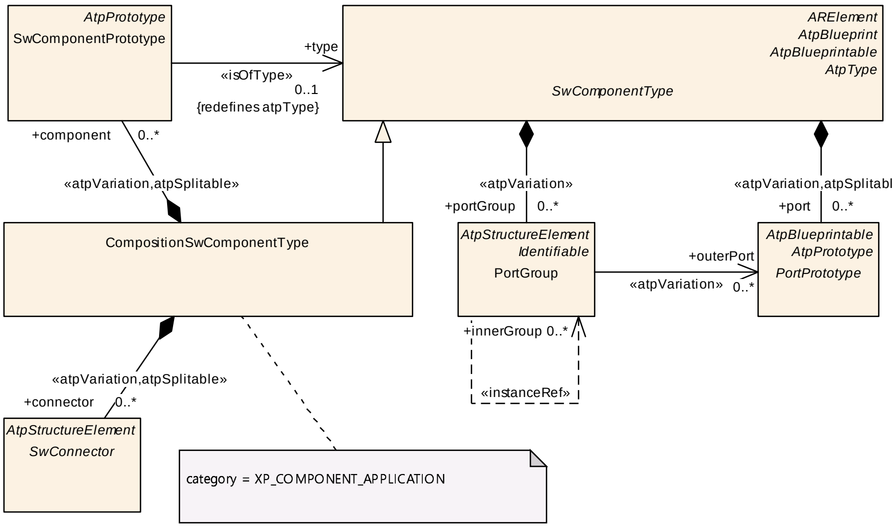

Figure 3.3: Modeling of Abstract Platform Components

Since CompositionSwComponentType is used in this way instead of a designated AtomicSwComponentType, it means CompositionSwComponentType is used all the way down the component hierarchy tree. To distinguish between the cases where a CompositionSwComponentType is designated as an actual plain composite software component, or as a 'quasi' atomic software component the category is restricted depending on the intent.

Without [constr_6803], it is very arbitrary how to trace the usage of a component between an XP and a concrete platform - foreseeably the abstract component could only be derived by default to say an arbitrary representation in a downstream platform and it would be a pure manual step and not allow for any future automation. Usage of the category should therefore allow an architect to specify some finer detailing of the component type.

[TPS_APSD_01005]{DRAFT} **Identification of component types in an abstract platform** ⌈The abstract platform uses the category of the CompositionSwComponentType as a means to optionally identify the intended usage of the CompositionSwComponentType.⌋*()*

[constr_6803]{DRAFT} **Standardized values of CompositionSwComponentType.category** ⌈In a System with the category set to ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION, any CompositionSwComponentType which is referenced by a SwComponentPrototype in the role type shall have the category set to:

- XP_COMPONENT_APPLICATION

⌋*()*

[TPS_APSD_01020]{DRAFT} **Semantics of a CompositionSwComponentType of category XP_COMPONENT_APPLICATION** ⌈A composition of category XP_COMPONENT_APPLICATION in an abstract platform represents an application software component.⌋*()*

See A.2 for an example ARXML listing.

Class	**CompositionSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	A CompositionSwComponentType aggregates SwComponentPrototypes (that in turn are typed by SwComponentTypes) as well as SwConnectors for primarily connecting SwComponentPrototypes among each others and towards the surface of the CompositionSwComponentType. By this means, hierarchical structures of software-components can be created. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

component	SwComponentPrototype	*	aggr	**Stereotypes:** atpSplitable; atpVariation
				Tags:
				atp.Splitkey=component.shortName, component.variationPoint.shortLabel
				vh.latestBindingTime=postBuild

connector	SwConnector	*	aggr	SwConnectors have the principal ability to establish a connection among PortPrototypes. They can have many roles in the context of a CompositionSwComponentType. Details are refined by subclasses. The aggregation of SwConnectors is subject to variability with the purpose to support variant data flow. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=connector.shortName, connector.variationPoint.shortLabel vh.latestBindingTime=postBuild

Table 3.3: CompositionSwComponentType

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

Table 3.4: SwComponentPrototype

Class	***SwComponentType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for AUTOSAR software components.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AtomicSwComponentType*, CompositionSwComponentType, ParameterSwComponentType			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***SwComponentType*** (abstract)			
port	PortPrototype	*	aggr	The PortPrototypes through which this SwComponentType can communicate. The aggregation of PortPrototype is subject to variability with the purpose to support the conditional existence of PortPrototypes. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=port.shortName, port.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
portGroup	PortGroup	*	aggr	A port group being part of this component. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

swComponentDocumentation	SwComponentDocumentation	0..1	aggr	This adds a documentation to the SwComponentType.
				Stereotypes: atpSplitable; atpVariation
				Tags:
				atp.Splitkey=swComponentDocumentation, swComponentDocumentation.variationPoint.shortLabel
				vh.latestBindingTime=preCompileTime
				xml.sequenceOffset=-10

Table 3.5: SwComponentType

3.3.4 Connectors

While support for modeling of port connectors in an XP entirely makes sense for certain downstream architectures, in others it doesn't. Especially for SOA based platforms it can be argued that they are superfluous - SOA middlewares typically only create the "connection" when the provided service is "found" during run time after the other side has initiated a search.

[TPS_APSD_01012]{DRAFT} **Modeling of connectors in an abstract platform** ⌈The XP allows modeling of connectors, but defers their concrete application to a downstream platform.⌋*()*

In other words, the XP is agnostic of the concrete platform, but to facilitate a usage of connectors in a concrete platform where they have real semantics, it does not prohibit their use.

The XP therefore takes over the modeling of connectors from [6] chapter "Overview::Composition::Connectors".

3.3.5 Port Groups

Port grouping is fairly standard in component models, though it is really at the discretion of the model itself what the semantic meaning of a port group is. Several scenarios are possible such as limiting inclusion of discrete ports in discrete groups or allowing discrete ports to be mapped into different groups. Some models define an abstract port group as being a composition which may be further decomposed in a downstream platform.

[TPS_APSD_01009]{DRAFT} **Grouping of ports in an abstract platform** ⌈Assigning discrete ports to zero or more port groups shall be possible in an abstract platform.⌋*()*

3.4 Port Interfaces

3.4.1 Overview

The XP follows the same general principles laid down in [6] chapter "Overview::Port Interface...". The XP restricts the model to disallow that the same port is read/write.

[TPS_APSD_01007]{DRAFT} **Prototyping of ports in an abstract platform** ⌈An abstract platform port is either in the role of requirer or provider but not both.⌋*()*

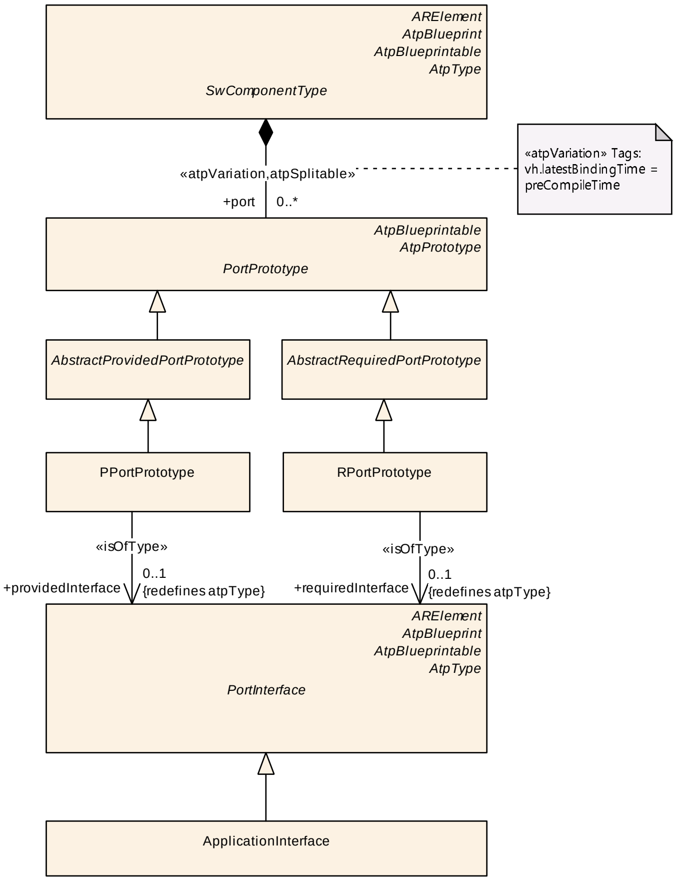

Figure 3.4: Modeling of Abstract Platform Ports

3.4.2 Port Interfaces

The XP does not mandate for specific types of PortInterfaces. Whereas in CP/AP, the respective VFB models in each, specify specific types of PortInterfaces for an intended functional usage, e.g.: AP ServiceInterface is intended for a SOA based deployment, the XP opts for an interface type which could be applied generically.

The XP does allow to provide some further semantics to a ApplicationInterface to indicate an intended usage for a certain port via ApplicationInterface.category. This serves as a hint which may be optionally considered when deriving (if it has a semantic meaning on the downstream platform), even though the ApplicationInterface itself does convey anything relating to the functional usage or underlying signaling architecture between ports [TPS_APSD_01010].

[TPS_APSD_01008]{DRAFT} **Generic typing of interfaces in an abstract platform** ⌈The abstract platform does not semantically bind types of PortInterfaces to a particular functional usage.⌋*()*

[TPS_APSD_01010]{DRAFT} **Agnosticism of abstract platform interfaces to middleware deployments** ⌈An abstract platform interface is agnostic of both architecture and any middleware deployment options.⌋*()*

[TPS_APSD_01022]{DRAFT} **Semantics of a ApplicationInterface** ⌈A ApplicationInterface inherits from a PortInterface and provides an functionally agnostic PortInterface type to allow data exchange between PortPrototypes.⌋ *()*

[constr_6807]{DRAFT} **Exclusivity of a ApplicationInterface to an Abstract Platform** ⌈A ApplicationInterface shall not type a PortPrototype unless the category of the System is ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION.⌋*()*

The rationale for [constr_6807] is grounded in [TPS_APSD_01003].

3.4.2.1 Elements of Application Interface

The make-up of a ApplicationInterface borrows from the approach taken in the AP ServiceInterface to allow flexibility as to the choice of how data is exchanged.[1]

It is possible that a downstream platform only supports atomic interface types, in this case, during derivation, the individual elements of a ApplicationInterface must be mapped to discrete atomic interfaces. Obviously, this may have an impact on ports which would then need to be created or alternatively some facade pattern employed.

[TPS_APSD_01023]{DRAFT} **Elements of a ApplicationInterface** ⌈A ApplicationInterface allows the following forms of data exchange:

[1] An alternative approach to a ApplicationInterface could be to use CP style atomic interfaces (a singular message exchange element). However, the aggregation of singular exchange elements in a ApplicationInterface offers more flexibility.

- ClientServerOperation aggregated in the role command.
- VariableDataPrototype aggregated in the role indication.
- Field aggregated in the role attribute.

⌋*()*

[TPS_APSD_01024]{DRAFT} **Semantics of a ApplicationInterface.command** ⌈A command is a RPC with optional function arguments, called by the requirer (client) and executed on the side of the provider (server).⌋*()*

[TPS_APSD_01025]{DRAFT} **Semantics of a ApplicationInterface.indication** ⌈An indication is a plain block of data that shall be updated (indicated) by the provider (server).⌋*()*

[TPS_APSD_01039]{DRAFT} **Semantics of a ApplicationInterface.attribute** ⌈An attribute (analogous to a "field" in a class/object) offers the combined features of a command and indication. A requirer (client) side entity may get or set an attribute or, receive an indication from a provider (server) side entity when the value of the attribute has been updated (irrespective of whether the value has changed from the previous value).⌋*()*

[TPS_APSD_01040]{DRAFT} **Attributes of a Field** ⌈A Field supports the following attributes with associated semantics:

- a hasGetter: setting hasGetter to TRUE/FALSE is equivalent to allowing/forbidding requirer (client) side retrieval of the attribute value.
- a hasSetter: setting hasSetter to TRUE/FALSE is equivalent to allowing/forbidding requirer (client) side updating of the attribute value.
- a hasNotifier: setting hasNotifier to TRUE/FALSE is equivalent to allowing/forbidding the provider (server) to send notification updates to the requirer (client) when the attribute value has changed (irrespective of whether the value has changed from the previous value).

⌋*()*

[constr_6806]{DRAFT} **Standardized values of ApplicationInterface.category** ⌈The category of a ApplicationInterface can be set to either:

- XP_PORT_SECURITY
- XP_PORT_TIMESYNC
- XP_PORT_STORAGE
- XP_PORT_APPLICATION
- XP_PORT_SAFETY

⌋*()*

[TPS_APSD_01026]{DRAFT} **Semantics of a ApplicationInterface of category XP_PORT_SECURITY** ⌈A ApplicationInterface of category XP_PORT_SECURITY represents a control port to a security entity: e.g. a cryptographic or authentication entity.⌋*()*

[TPS_APSD_01027]{DRAFT} **Semantics of a ApplicationInterface of category XP_PORT_TIMESYNC** ⌈A ApplicationInterface of category XP_PORT_TIMESYNC represents a control port to a time synchronization entity: e.g. AP AbstractSynchronizedTimeBaseInterface.⌋*()*

[TPS_APSD_01028]{DRAFT} **Semantics of a ApplicationInterface of category XP_PORT_STORAGE** ⌈A ApplicationInterface of category XP_STORAGE represents a port to a storage entity used to hold persistent data: e.g. AP PersistencyInterface or CP NvDataInterface.⌋*()*

[TPS_APSD_01029]{DRAFT} **Semantics of a ApplicationInterface of category XP_PORT_APPLICATION** ⌈A ApplicationInterface of category XP_PORT_APPLICATION represents a general application data port: e.g. an AP ServiceInterface.⌋*()*

[TPS_APSD_01038]{DRAFT} **Semantics of a ApplicationInterface of category XP_PORT_SAFETY** ⌈A ApplicationInterface of category XP_PORT_APPLICATION represents a port to a safety entity: e.g. an AP PlatformHealthManagementInterface or an equivalent CP entity.⌋*()*

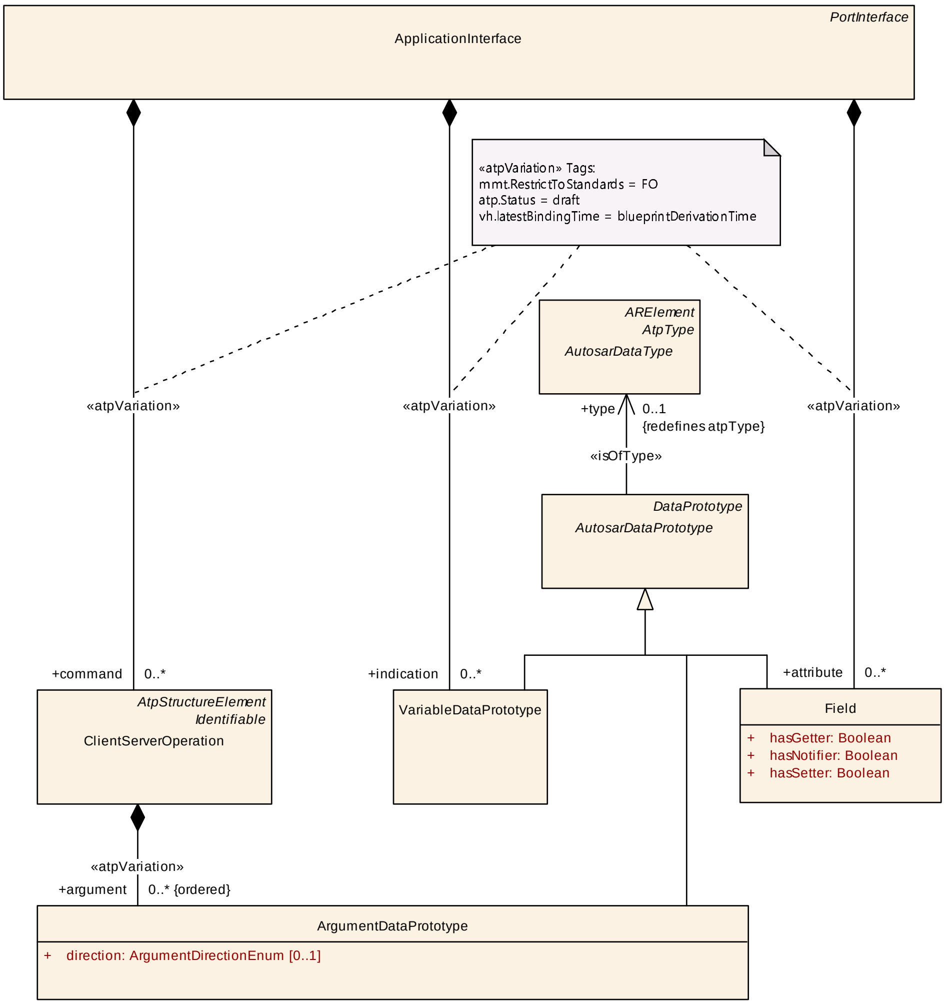

Figure 3.5: Modeling of Abstract Platform interfaces

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*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	ApplicationInterface, ClientServerInterface, *DataInterface*, ModeSwitchInterface, TriggerInterface			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 3.6: PortInterface

Class	**ApplicationInterface**
Package	M2::AUTOSARTemplates::AbstractPlatform

▽

△

Class	**ApplicationInterface**			

Note	This represents the ability to define a PortInterface that consists of a composition of commands (method calls), indications (events) and attributes (fields)
	Tags:
	atp.Status=draft
	atp.recommendedPackage=Interfaces			

Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
attribute	Field	*	aggr	This represents the set of attributes defined in the context of an Abstract Platform ApplicationInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
command	ClientServerOperation	*	aggr	This represents the collection of commands or function calls (with optional data arguments) defined in the context of an ApplicationInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
indication	VariableDataPrototype	*	aggr	This represents the collection of indication or events (with optional data argument) defined in the context of an ApplicationInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime

Table 3.7: ApplicationInterface

Class	**ClientServerOperation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A remote procedure call declared within the scope of the current interface.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
argument (ordered)	ArgumentDataPrototype	*	aggr	An argument of this ClientServerOperation **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=blueprintDerivationTime

Table 3.8: ClientServerOperation

Class	**VariableDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	A VariableDataPrototype is used to contain arbitrary values in a software component. In particular, the value of a VariableDataPrototype is likely to change over its lifetime.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**VariableDataPrototype**			
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype

Table 3.9: VariableDataPrototype

Class	**ArgumentDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *Multilanguage Referrable*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
direction	ArgumentDirection Enum	0..1	attr	This attribute specifies the direction of the argument prototype.

Table 3.10: ArgumentDataPrototype

3.5 Data Types

3.5.1 Overview

The XP *partially* takes over the AUTOSAR data typing model and principles defined in [6] chapter "Data Description". With reference to [TPS_SWCT_01229] and the table "Abstraction Levels for Describing Data", only the *Application Data Level* shall be used.

The XP is concerned with a modeling of high-level data types and attributes of data types like the physical meaning of a data type. XP data types are not concerned with implementation or platform level data types; it is expected that these are fully in the domain of a concrete platform.

[TPS_APSD_01014]{DRAFT} **Allowed data types in an abstract platform** ⌈The abstract platform allows deferral of data typing or explicit data typing using:

- integrals in the form of ApplicationDataType.category=VALUE
- structures in the form of ApplicationDataType.category=STRUCTURE
- arrays in the form of ApplicationDataType.category=ARRAY
- strings in the form of ApplicationDataType.category=STRING
- booleans in the form of ApplicationDataType.category=BOOLEAN

⌋*()*

[TPS_APSD_01030]{DRAFT} **Exclusion of implementation level data types** ⌈The abstract platform does not support modeling of ImplementationDataType.⌋*()*

3.5.2 Properties of Data Definitions

The properties of data definitions from [6] chapter "Data Description::Properties of Data Definitions" also apply in XP. However, due to the reduced subset of supported categorys of ApplicationDataTypes (see 3.5.3), the list of SwDataDefProps attributes is therefore also constrained respectively.

The semantic meaning of those attributes defined in Table 3.11 is specified in [6] chapter "Data Description::Elements used in Properties of Data Definitions".

[constr_6812]{DRAFT} **SwDataDefProps applicable to ApplicationDataTypes exclusive to the abstract platform** ⌈A complete list of the allowed SwDataDefProps attributes and their multiplicities which are allowed for a given category is shown in table 3.11.⌋*()*

Attributes of SwDataDefProps	Root Elem.				Attribute Existence per Category				
	ApplicationDataType	ApplicationDeferredDataType	ApplicationRecordElement	ApplicationArrayElement	VALUE	STRUCTURE	ARRAY	STRING	BOOLEAN
annotation	x	x	x	x	*	*	*	*	*
compuMethod	x				0..1				0..1
dataConstr.dataConstrRule.physConstrs	x		x	x	0..1		0..1		0..1
dataConstr.dataConstrRule.internalConstrs	x		x	x	d/c[2]		d/c		d/c
displayFormat	x		x	x	0..1		0..1	0..1	0..1
invalidValue	x				0..1			0..1	0..1
swTextProps	x							1	
unit	x				0..1			0..1	0..1
Other Attributes below the Root Element									
element: ApplicationRecordElement	x		x	x		1..*			
element: ApplicationArrayElement	x		x	x			1		
ApplicationArrayElement.arraySizeSemantics	x						0..1		
ApplicationArrayElement.maxNumberOfElements	x						1		

Table 3.11: Allowed Attributes vs. category for ApplicationDataTypes

[2] don't care

Class	<<atpVariation>> **SwDataDefProps**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. **Tags:**vh.latestBindingTime=codeGenerationTime			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object.
				Tags:
				xml.roleElement=true
				xml.roleWrapperElement=true
				xml.sequenceOffset=20
				xml.typeElement=false
				xml.typeWrapperElement=false

baseType	SwBaseType	0..1	ref	Base type associated with the containing data object. **Tags:**xml.sequenceOffset=50
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:**xml.sequenceOffset=180
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:**xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:**xml.sequenceOffset=210
display Presentation	DisplayPresentation Enum	0..1	attr	This attribute controls the presentation of the related data for measurement and calibration tools.
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:**xml.sequenceOffset=255

swComparison Variable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process.
				Tags:
				xml.sequenceOffset=170
				xml.typeElement=false

swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects.
				Tags:
				xml.sequenceOffset=220
				xml.typeElement=false

swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:**xml.sequenceOffset=120
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:**xml.sequenceOffset=350

Table 3.12: SwDataDefProps

3.5.3 Data Type Categories

The basis for allowed application data types in an XP are those application data types cited in AUTOSAR Software-Component Template [6] chapter "Data Types::Data Categories" - but not all categorys of ApplicationDataType are supported in XP.

[constr_6810]{DRAFT} **Applicable categories for data types in an abstract platform** ⌈Table 3.13 defines the applicable data type categorys relating to applicable meta-model classes.⌋*()*

Category	Applicable to ...							Description
	ApplicationDataType	ApplicationDeferredDataType	ApplicationArrayDataType	ApplicationRecordDataType	ApplicationPrimitiveDataType	ApplicationRecordElement	ApplicationArrayElement	
VALUE					x	x	x	Contains a single value.
STRUCTURE				x		x	x	Holds one or several further elements which can have different AutosarDataTypes.
STRING					x	x	x	Contains a single value interpreted as a text string (note that it appears as a single value for the application domain).
ARRAY			x			x	x	A fixed-sized array of sub-elements of the same type.
BOOLEAN					x	x	x	Contains a single boolean (true/false) state.

Table 3.13: Usage of category for Data Types

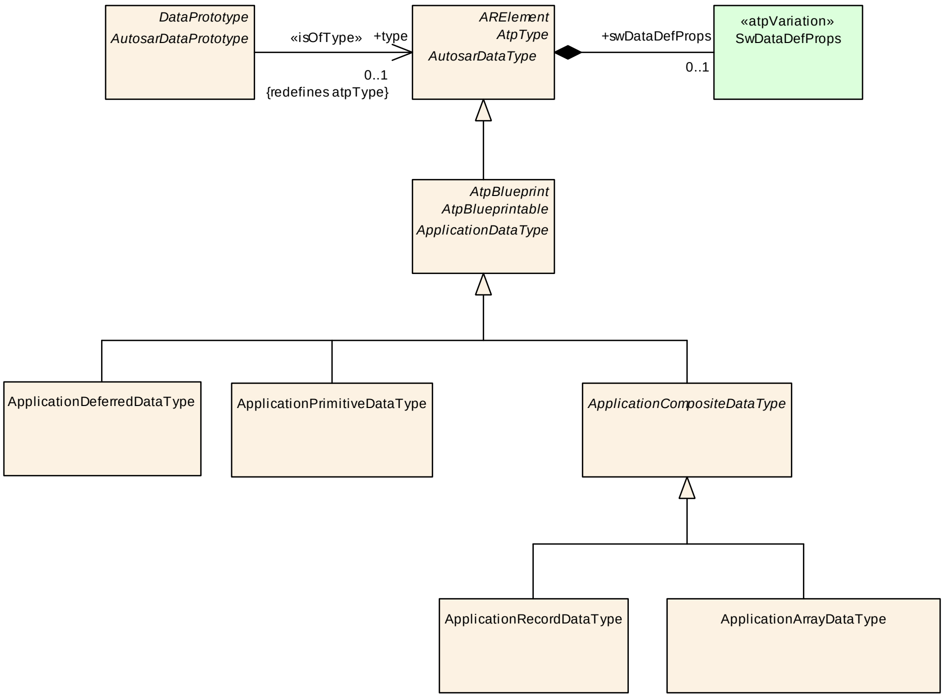

Figure 3.6: Modeling of Abstract Platform data types

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 ApplicationData Types only.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*ApplicationCompositeDataType*, ApplicationDeferredDataType, ApplicationPrimitiveDataType			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 3.14: 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	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 3.15: ApplicationPrimitiveDataType

Class	*ApplicationCompositeDataType* (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	Abstract base class for all application data types composed of other data types.			
Base	*ARElement*, *ARObject*, *ApplicationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	ApplicationArrayDataType, ApplicationRecordDataType			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 3.16: ApplicationCompositeDataType

Class	**ApplicationRecordDataType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	An application data type which can be decomposed into prototypes of other application data types. **Tags:**atp.recommendedPackage=ApplicationDataTypes			
Base	*ARElement*, *ARObject*, *ApplicationCompositeDataType*, *ApplicationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
element (ordered)	ApplicationRecordElement	*	aggr	Specifies an element of a record. The aggregation of ApplicationRecordElement is subject to variability with the purpose to support the conditional existence of elements inside a ApplicationrecordDataType. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table 3.17: ApplicationRecordDataType

Class	**ApplicationArrayDataType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	An application data type which is an array, each element is of the same application data type. **Tags:**atp.recommendedPackage=ApplicationDataTypes			
Base	*ARElement*, *ARObject*, *ApplicationCompositeDataType*, *ApplicationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
dynamicArraySizeProfile	String	0..1	attr	Specifies the profile which the array will follow if it is a variable size array.
element	ApplicationArrayElement	0..1	aggr	This association implements the concept of an array element. That is, in some cases it is necessary to be able to identify single array elements, e.g. as input values for an interpolation routine.

Table 3.18: ApplicationArrayDataType

3.5.4 Application Data Types

The XP is targeting a platform independent design of data exchange between SWCs (applications). To keep to the underlying goal of remaining independent of concrete platform implementation details, a description of the used data types in the XP is therefore naturally limited to application data types.

The AUTOSAR data type model starts with AutosarDataType. The meta-class AutosarDataType inherits from Identifiable which provides the identifying attributes needed: longName, shortName. The category is then used to identify the underlying category of application level data type.

3.5.5 Sub-classes of ApplicationDataType

The XP supports the sub-classes in the table in [constr_6810]. Partially those sub-classes are re-used from [6] chapter "Data Types::Application Data Type" and are thus defined there. The sub-classes defined purely by the XP are detailed here.

[TPS_APSD_01031]{DRAFT} **Sub-classes of ApplicationDataType** ⌈In an abstract platform, the abstract meta-class ApplicationDataType is sub-classed into:

- ApplicationDeferredDataType

⌋*()*

These XP specific sub-classes are detailed in the following sections.

3.5.5.1 Deferred Data Type

Due to the fact that a data type may not *yet* be known in the XP, or shall be defined later in the design in a downstream stage, XP typing can be deferred with the proviso that it shall be concretely defined during derivation to a concrete platform or mapping to a implementation data type.

This is done using the XP exclusive type called ApplicationDeferredDataType. Fully usable in an XP, together with their properties (Table 3.11).

[TPS_APSD_01015]{DRAFT} **Deferral of the category of data type in an abstract platform** ⌈The abstract platform provides a non-committal data type ApplicationDeferredDataType to allow deferral of an actual data type to a later stage.⌋*()*

[TPS_APSD_01032]{DRAFT} **Semantics of an ApplicationDeferredDataType** ⌈An ApplicationDeferredDataType represents a placeholder, Identifiable within a model, but having no actual applicable category of data type.⌋*()*

As mentioned previously in 3.5.3, AUTOSAR ApplicationDataTypes are assigned a category value from the Table 3.13. The ApplicationDeferredDataType however is an exception to this rule since it has no concrete type yet.

It is therefore necessary to exclude any assignment of categorys of type in a model [constr_6814]. Further to that, no properties of data definitions are assigned to ApplicationDeferredDataType which would convey in any way concrete data type characteristics [constr_6812]. In other words, the list of attributes is deliberately very constrained in order to be agnostic of concrete data typing.

[constr_6814]{DRAFT} **Restriction of ApplicationDeferredDataType.category** ⌈The category of an ApplicationDeferredDataType shall be unassigned/undefined.⌋*()*

Class	**ApplicationDeferredDataType**			
Package	M2::AUTOSARTemplates::AbstractPlatform			

Note	An placeholder data type in which the precise application data type is deferred to a later stage.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=ApplicationDataTypes			

Base	*ARElement*, *ARObject*, *ApplicationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 3.19: ApplicationDeferredDataType

3.5.6 Type Tracing

As mentioned previously, the XP is not concerned with those data types below the level of ApplicationDataTypes. Data type tracing between an XP and a concrete platform model must be done on the same level - in the context of AUTOSAR, that means tracing only between ApplicationDataTypes in XP/CP/AP as shown in Figure 3.7.

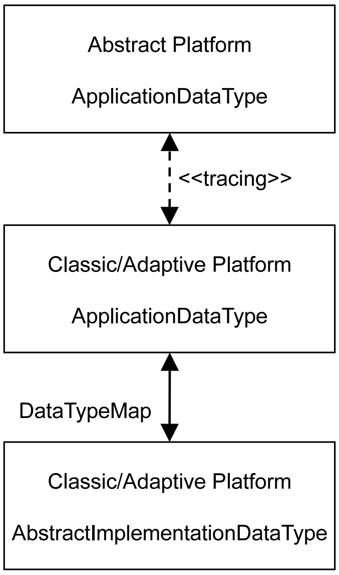
Abstract Platform
ApplicationDataType
<<tracing>>
Classic/Adaptive Platform
ApplicationDataType
DataTypeMap
Classic/Adaptive Platform
AbstractImplementationDataType

Figure 3.7: Permitted data type tracing

Any usage of XP ApplicationDataTypes in a CP or AP is not allowed (Figure 3.7 *left*). Any indirect tracing between XP ApplicationDataTypes and either CP AbstractImplementationDataTypes or AP AbstractImplementationDataTypes is not supported (Figure 3.7 *right*).

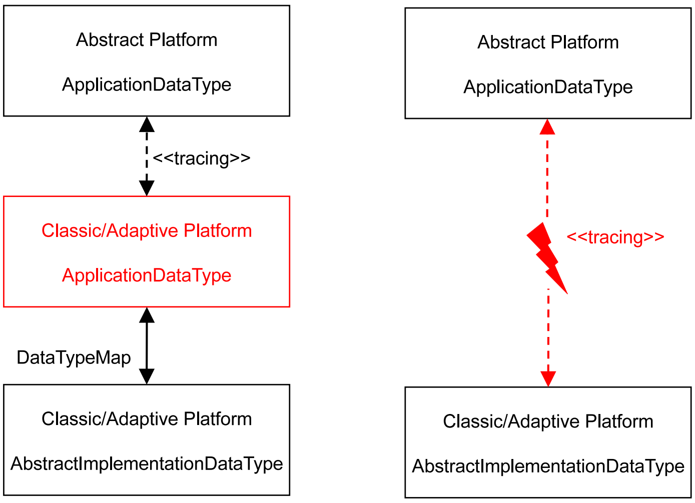
Abstract Platform
ApplicationDataType
<<tracing>>
Classic/Adaptive Platform
ApplicationDataType
DataTypeMap
Classic/Adaptive Platform
AbstractImplementationDataType
Abstract Platform
ApplicationDataType
<<tracing>>
Classic/Adaptive Platform
AbstractImplementationDataType

Figure 3.8: Prohibited data type tracing

[TPS_APSD_01036]{DRAFT} **Data type tracing between abstract and concrete platform models** ⌈Tracing of data types between abstract and concrete platform models is solely on the level of ApplicationDataTypes.⌋*()*

For a model based example view on tracing see chapter A.3. The remainder of this section details any aspects to consider when tracing (deriving/abstracting) XP specific sub-classes of ApplicationDataTypes (listed by sub-chapter here).

3.5.6.1 Deferred Data Type

Since an ApplicationDeferredDataType is basically a placeholder type, and holds no concrete data type properties, it is straightforward to trace this type between an abstract and concrete platform.

[TPS_APSD_01037]{DRAFT} **Compatibility of an ApplicationDeferredDataType** ⌈During tracing, the ApplicationDeferredDataType provides *none*, and the concrete platform type provides *all* of the aspects of necessary typing.⌋*()*

[TPS_APSD_01016]{DRAFT} **Concrete data type resolution of an ApplicationDeferredDataType** ⌈The precise data typing of a ApplicationDeferredDataType is not required until the methodology step before, or latest during:

- derivation of ApplicationDataTypes defined in the context of an XP, to corresponding ApplicationDataTypes defined in the context of either a CP or an AP.

- derivation of ApplicationDataTypes defined in the context of an XP, to a corresponding domain specific representation in a non-AUTOSAR platform.

⌋()

[TPS_APSD_01033]{DRAFT} **Traceability of an ApplicationDeferredDataType** ⌈If the concrete platform is:

- an AUTOSAR platform: an ApplicationDeferredDataType can be traced to any of the supported ApplicationDataTypes on the concrete platform.
- a non-AUTOSAR platform: tracing is domain specific.

⌋()

4 Requirements

4.1 Overview

The AUTOSAR meta-model already provides a healthy set of meta-classes for the topic of requirements in the AUTOSAR Standardization Template [2] [TPS_STDT_00060]. For requirements engineering (annotation, documentation, rationalization, traceability) in an XP, they can be directly applied.

The XP allows requirement engineering to be performed within the context of an XP system description. A top-level requirement can be added which can be recursively broken-down (decomposed) into N x *child* level requirements and annotated to an XP description.

It is at the discretion of the designer how and when to do this step and to decide when the current decomposition level is sufficient. During the concrete platform implementation stage a developer would then implement according to the requirements.There are no restrictions on what a requirement is, nor on the number of decompositions of a requirement. The meta-class StructuredReq may be reused directly for requirement specification.

[TPS_APSD_01034]{DRAFT} **Requirement annotation and in an abstract platform** ⌈An abstract platform description supports recursive depths of requirements annotation, decomposition.⌋*()*

For a detailed description of AUTOSAR's support for traceability of all kinds refer to [5] chapter "Documentation Support".

A Examples

This chapter contains a collection of examples that reflect concepts described in different chapters of this document. The content of the chapter provides mere explanation and does not add anything to the model semantics.

A.1 System

The listing in A.1 illustrates the definition of a System to describe an abstract platform.

Listing A.1: Example ARXML for abstract platform system

```
<SYSTEM>
  <SHORT-NAME>MySystem</SHORT-NAME>
  <CATEGORY>ABSTRACT_PLATFORM_SYSTEM_DESCRIPTION</CATEGORY>
  <ROOT-SOFTWARE-COMPOSITIONS>
    <ROOT-SW-COMPOSITION-PROTOTYPE>
    <SHORT-NAME>MyRootSwComposition</SHORT-NAME>
    <SOFTWARE-COMPOSITION-TREF DEST="COMPOSITION-SW-COMPONENT-TYPE">/
        ARDesign/SystemDesign/MyTopLevelComposition</SOFTWARE-COMPOSITION-
        TREF>
    </ROOT-SW-COMPOSITION-PROTOTYPE>
  </ROOT-SOFTWARE-COMPOSITIONS>
  <SYSTEM-VERSION>0.1.0</SYSTEM-VERSION>
</SYSTEM>
```

A.2 Component hierarchy

The listing in A.2 illustrates the usage of CompositionSwComponentType to define a hierarchy of components.

Listing A.2: Example ARXML for abstract software components

```
<AR-PACKAGE>
  <SHORT-NAME>Components</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>RadarFusionUnit</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>components</SHORT-NAME>
          <ELEMENTS>
            <COMPOSITION-SW-COMPONENT-TYPE>
              <SHORT-NAME>Unit</SHORT-NAME>
              <COMPONENTS>
                <SW-COMPONENT-PROTOTYPE>
                  <SHORT-NAME>radar</SHORT-NAME>
                  <TYPE-TREF DEST="COMPOSITION-SW-COMPONENT-TYPE">/ARDesign
                      /Components/RadarFusionUnit/components/UnitRadar</TYPE
                      -TREF>
                </SW-COMPONENT-PROTOTYPE>
                <SW-COMPONENT-PROTOTYPE>
                  <SHORT-NAME>camera</SHORT-NAME>
                  <TYPE-TREF DEST="COMPOSITION-SW-COMPONENT-TYPE">/ARDesign
                     /Components/RadarFusionUnit/components/UnitCamera</
                     TYPE-TREF>
                </SW-COMPONENT-PROTOTYPE>
              </COMPONENTS>
            </COMPOSITION-SW-COMPONENT-TYPE>
            <COMPOSITION-SW-COMPONENT-TYPE>
              <SHORT-NAME>UnitRadar</SHORT-NAME>
              <COMPONENTS>
                <SW-COMPONENT-PROTOTYPE>
                  <SHORT-NAME>app</SHORT-NAME>
                  <TYPE-TREF DEST="COMPOSITION-SW-COMPONENT-TYPE">/ARDesign
                     /Components/RadarFusionUnit/components/UnitRadarApp</
                     TYPE-TREF>
                </SW-COMPONENT-PROTOTYPE>
              </COMPONENTS>
            </COMPOSITION-SW-COMPONENT-TYPE>
            <COMPOSITION-SW-COMPONENT-TYPE>
              <SHORT-NAME>UnitCamera</SHORT-NAME>
              <CATEGORY>XP_COMPONENT_APPLICATION</CATEGORY>
              <COMPONENTS>
                <SW-COMPONENT-PROTOTYPE>
                  <SHORT-NAME>app</SHORT-NAME>
                  <TYPE-TREF DEST="COMPOSITION-SW-COMPONENT-TYPE">/ARDesign
                     /Components/RadarFusionUnit/components/UnitCameraApp</
                     TYPE-TREF>
                </SW-COMPONENT-PROTOTYPE>
                <SW-COMPONENT-PROTOTYPE>
                  <SHORT-NAME>sensor</SHORT-NAME>
                  <TYPE-TREF DEST="COMPOSITION-SW-COMPONENT-TYPE">/ARDesign
                     /Components/RadarFusionUnit/components/
                     UnitCameraSensor</TYPE-TREF>
                </SW-COMPONENT-PROTOTYPE>
              </COMPONENTS>
            </COMPOSITION-SW-COMPONENT-TYPE>
            <COMPOSITION-SW-COMPONENT-TYPE>
              <SHORT-NAME>UnitCameraSensor</SHORT-NAME>
              <CATEGORY>XP_COMPONENT_APPLICATION</CATEGORY>
            </COMPOSITION-SW-COMPONENT-TYPE>
            <COMPOSITION-SW-COMPONENT-TYPE>
              <SHORT-NAME>UnitCameraApp</SHORT-NAME>
              <CATEGORY>XP_COMPONENT_APPLICATION</CATEGORY>
            </COMPOSITION-SW-COMPONENT-TYPE>
            <COMPOSITION-SW-COMPONENT-TYPE>
              <SHORT-NAME>UnitRadarApp</SHORT-NAME>
              <CATEGORY>XP_COMPONENT_APPLICATION</CATEGORY>
            </COMPOSITION-SW-COMPONENT-TYPE>
          </ELEMENTS>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

A.3 Data type tracing

The model example A.1 illustrates a meta-model view on tracing between an `XP` and a `CP`/`AP`.

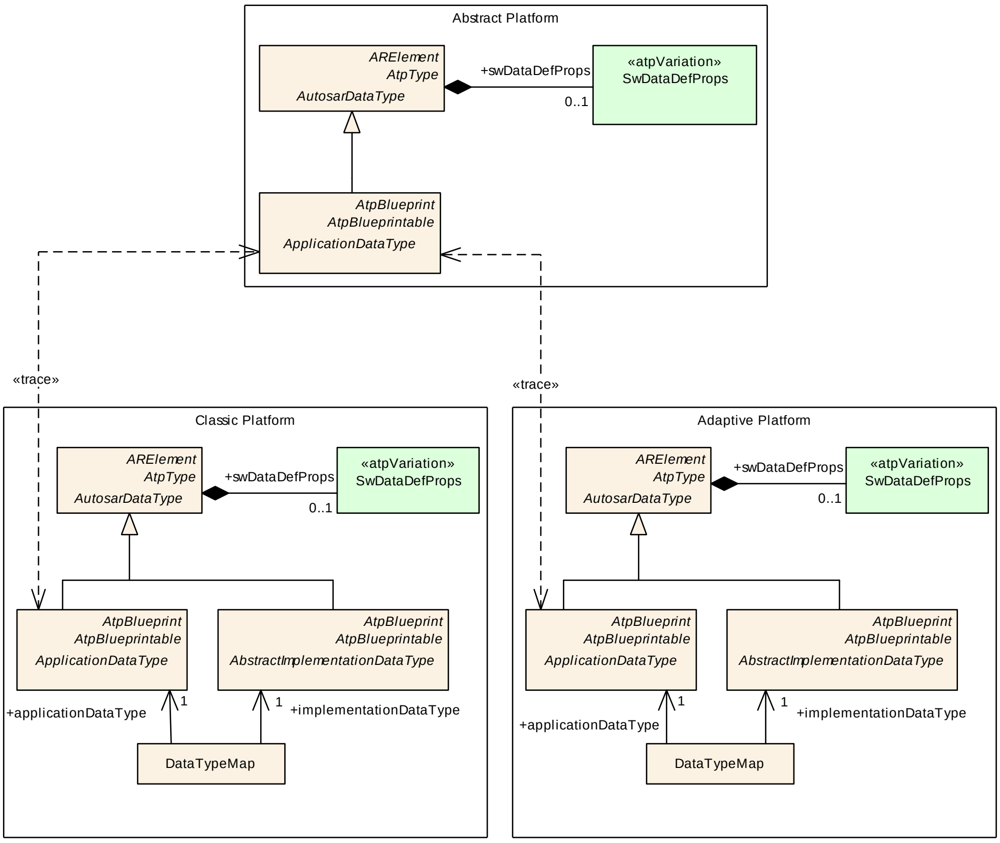

Figure A.1: Data type tracing

B 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	**AUTOSAR**			
Package	M2::AUTOSARTemplates::AutosarTopLevelStructure			
Note	Root element of an AUTOSAR description, also the root element in corresponding XML documents. **Tags:**xml.globalElement=true			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
adminData	AdminData	0..1	aggr	This represents the administrative data of an Autosar file. **Tags:**xml.sequenceOffset=10

arPackage	ARPackage	*	aggr	This is the top level package in an AUTOSAR model.
				Stereotypes: atpSplitable; atpVariation
				Tags:
				atp.Splitkey=arPackage.shortName, arPackage.variationPoint.shortLabel
				vh.latestBindingTime=blueprintDerivationTime
				xml.sequenceOffset=30

fileInfoComment	FileInfoComment	0..1	aggr	This represents a possibility to provide a structured comment in an AUTOSAR file.
				Stereotypes: atpStructuredComment
				Tags:
				xml.roleElement=true
				xml.sequenceOffset=-10
				xml.typeElement=false

introduction	DocumentationBlock	0..1	aggr	This represents an introduction on the Autosar file. It is intended for example to represent disclaimers and legal notes. **Tags:**xml.sequenceOffset=20

Table B.1: AUTOSAR

Class	***AbstractImplementationDataType*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	This meta-class represents an abstract base class for different flavors of ImplementationDataType.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	ImplementationDataType			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table B.2: AbstractImplementationDataType

Class	**ApplicationArrayElement**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes
Note	Describes the properties of the elements of an application array data type.

▽

△

Class	**ApplicationArrayElement**			
Base	*ARObject*, *ApplicationCompositeElementDataPrototype*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arraySizeHandling	ArraySizeHandlingEnum	0..1	attr	The way how the size of the array is handled.
arraySizeSemantics	ArraySizeSemanticsEnum	0..1	attr	This attribute controls how the information about the array size shall be interpreted.
indexDataType	ApplicationPrimitiveDataType	0..1	ref	This reference can be taken to assign a CompuMethod of category TEXTTABLE to the array. The texttable entries associate a textual value to an index number such that the element with that index number is represented by a symbolic name.
maxNumberOfElements	PositiveInteger	0..1	attr	The maximum number of elements that the array can contain. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table B.3: ApplicationArrayElement

Class	**ApplicationRecordElement**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Describes the properties of one particular element of an application record data type.			
Base	*ARObject*, *ApplicationCompositeElementDataPrototype*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
isOptional	Boolean	0..1	attr	This attribute represents the ability to declare the enclosing ApplicationRecordElement as optional. This means the that, at runtime, the ApplicationRecordElement may or may not have a valid value and shall therefore be ignored. The underlying runtime software provides means to set the ApplicationRecordElement as not valid at the sending end of a communication and determine its validity at the receiving end.

Table B.4: ApplicationRecordElement

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*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Subclasses	ApplicationSwComponentType, ComplexDeviceDriverSwComponentType, EcuAbstractionSwComponentType, SensorActuatorSwComponentType, ServiceProxySwComponentType, ServiceSwComponentType			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***AtomicSwComponentType*** (abstract)			
internalBehavior	SwcInternalBehavior	0..1	aggr	The SwcInternalBehaviors owned by an AtomicSw ComponentType can be located in a different physical file. Therefore the aggregation is <<atpSplitable>>. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalBehavior.shortName, internal Behavior.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the AtomicSw ComponentType. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=symbolProps.shortName

Table B.5: AtomicSwComponentType

Class	***AutosarDataType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	Abstract base class for user defined AUTOSAR data types for software.			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Subclasses	*AbstractImplementationDataType*, *ApplicationDataType*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
swDataDef Props	SwDataDefProps	0..1	aggr	The properties of this AutosarDataType.

Table B.6: AutosarDataType

Class	**DataConstr**			
Package	M2::MSR::AsamHdo::Constraints::GlobalConstraints			
Note	This meta-class represents the ability to specify constraints on data. **Tags:**atp.recommendedPackage=DataConstrs			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

dataConstrRule	DataConstrRule	*	aggr	This is one particular rule within the data constraints.
				Tags:
				xml.roleElement=true
				xml.roleWrapperElement=true
				xml.sequenceOffset=30
				xml.typeElement=false
				xml.typeWrapperElement=false

Table B.7: DataConstr

Class	**DataConstrRule**			
Package	M2::MSR::AsamHdo::Constraints::GlobalConstraints			
Note	This meta-class represents the ability to express one specific data constraint rule.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**DataConstrRule**			
constrLevel	Integer	0..1	attr	This attribute describes the category of a constraint. One of its functions is in the area of constraint violation, where it can be used from a certain level, to produce error messages. The lower the level, the more stringent the check. Used to distinguish hard or soft limits. **Tags:**xml.sequenceOffset=20
internalConstrs	InternalConstrs	0..1	aggr	Describes the limitations applicable on the internal domain (as opposed to the physical domain). **Tags:**xml.sequenceOffset=40
physConstrs	PhysConstrs	0..1	aggr	Describes the limitations applicable on the physical domain (as opposed to the internal domain). **Tags:**xml.sequenceOffset=30

Table B.8: DataConstrRule

Class	**Field**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents the ability to define a piece of data that can be accessed with read and/or write semantics. It is also possible to generate a notification if the value of the data changes. **Tags:**atp.Status=draft			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
hasGetter	Boolean	1	attr	This attribute controls whether read access is foreseen to this field. **Tags:**atp.Status=draft
hasNotifier	Boolean	1	attr	This attribute controls whether a notification semantics is foreseen to this field. **Tags:**atp.Status=draft
hasSetter	Boolean	1	attr	This attribute controls whether write access is foreseen to this field. **Tags:**atp.Status=draft

Table B.9: Field

Class	***Identifiable*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*
Subclasses	ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstanceFilter*, *AbstractServiceInstance*, *AdaptiveModuleInstantiation*, ApplicationEndpoint, ApplicationError, ArtifactChecksum, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, BlockState, *BuildActionEntity*, BuildActionEnvironment, Chapter, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ▽

▽

△

Class	***Identifiable*** (abstract)			
	△ ConsumedEventGroup, CouplingPort, *CouplingPortStructuralElement*, CryptoKeySlot, *CryptoServiceMapping*, DataPrototypeGroup, DataTransformation, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, DiagnosticFunctionInhibitSource, *DiagnosticRoutineSubfunction*, DltApplication, DltArgument, DltMessage, DoIpInterface, DoIpLogicAddress, DoIpRoutingActivation, EndToEndProtection, EthernetWakeupSleepOnDatalineConfig, EventHandler, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FlexrayArTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, InternalTriggeringPoint, Keyword, LifeCycleState, Linker, MacMulticastGroup, McDataInstance, MemorySection, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, *NmNode*, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, ResourceConsumption, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecureCommunicationAuthenticationProps, SecureCommunicationFreshnessProps, SecurityEventContextProps, *ServiceNeeds*, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SomeipTpChannel, *SpecElementReference*, *StackUsage*, StaticSocketConnection, StructuredReq, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SystemMapping, *TimeBaseResource*, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, *TransformationProps*, TransformationTechnology, Trigger, VariableAccess, VariationPointProxy, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:**xml.sequenceOffset=-25
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:**xml.sequenceOffset=-50
desc	MultiLanguageOverviewParagraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:**xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:**xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a ▽

▽

△

Class	***Identifiable*** (abstract)			
				△ DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:**xml.attribute=true

Table B.10: Identifiable

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*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dynamicArraySizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.
isStructWithOptionalElement	Boolean	0..1	attr	This attribute is only valid if the attribute category is set to STRUCTURE. If set to True, this attribute indicates that the ImplementationDataType has been created with the intention to define at least one element of the structure as optional.
subElement (ordered)	ImplementationDataTypeElement	*	aggr	Specifies an element of an array, 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=symbolProps.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 B.11: ImplementationDataType

Class	***MultilanguageReferrable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders). They also may have a longName. But they are not considered to contribute substantially to the overall structure of an AUTOSAR description. In particular it does not contain other Referrables.			
Base	*ARObject*, *Referrable*			
Subclasses	Caption, DefItem, DocumentationContext, *Identifiable*, SdgCaption, *TraceReferrable*, *Traceable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
longName	MultilanguageLongName	0..1	aggr	This specifies the long name of the object. Long name is targeted to human readers and acts like a headline.

Table B.12: MultilanguageReferrable

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

Table B.13: PortPrototype

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

Table B.14: Referrable

Class	**StructuredReq**			
Package	M2::MSR::Documentation::BlockElements::RequirementsTracing			
Note	This represents a structured requirement. This is intended for a case where specific requirements for features are collected. Note that this can be rendered as a labeled list.			
Base	*ARObject*, *DocumentViewSelectable*, *Identifiable*, *MultilanguageReferrable*, *Paginateable*, *Referrable*, *Traceable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

appliesTo	standardNameEnum	*	attr	This attribute represents the platform the requirement is assigned to.
				Tags:
				xml.namePlural=APPLIES-TO-DEPENDENCIES
				xml.sequenceOffset=25

conflicts	DocumentationBlock	0..1	aggr	This represents an informal specification of conflicts. **Tags:**xml.sequenceOffset=40
date	DateTime	1	attr	This represents the date when the requirement was initiated. **Tags:**xml.sequenceOffset=5
dependencies	DocumentationBlock	0..1	aggr	This represents an informal specification of dependencies. Note that upstream tracing should be formalized in the property trace provided by the superclass Traceable. **Tags:**xml.sequenceOffset=30
description	DocumentationBlock	0..1	aggr	This represents the general description of the requirement. **Tags:**xml.sequenceOffset=10
importance	String	1	attr	This allows to represent the importance of the requirement. **Tags:**xml.sequenceOffset=8
issuedBy	String	1	attr	This represents the person, organization or authority which issued the requirement. **Tags:**xml.sequenceOffset=6
rationale	DocumentationBlock	0..1	aggr	This represents the rationale of the requirement. **Tags:**xml.sequenceOffset=20
remark	DocumentationBlock	0..1	aggr	This represents an informal remark. Note that this is not modeled as annotation, since these remark is still essential part of the requirement. **Tags:**xml.sequenceOffset=60
supporting Material	DocumentationBlock	0..1	aggr	This represents an informal specification of the supporting material. **Tags:**xml.sequenceOffset=50
testedItem	Traceable	*	ref	This association represents the ability to trace on the same specification level. This supports for example the of acceptance tests. **Tags:**xml.sequenceOffset=70
type	String	1	attr	This attribute allows to denote the type of requirement to denote for example is it an "enhancement", "new feature" etc. **Tags:**xml.sequenceOffset=7

▽

△

Class	**StructuredReq**			
useCase	DocumentationBlock	0..1	aggr	This describes the relevant use cases. Note that formal references to use cases should be done in the trace relation. **Tags:**xml.sequenceOffset=35

Table B.15: StructuredReq

C History of Constraints and Specification Items

Please note that the lists in this chapter also include constraints and specification items that have been removed from the specification in a later version. These constraints and specification items do not appear as hyperlinks in the document.

C.1 Constraint and Specification Item History of this document according to AUTOSAR Release R19-11

C.1.1 Added Traceables in R19-11

Number	Heading
[TPS_APSD_01000]	Principle of an abstract platform system description
[TPS_APSD_01001]	Modeling of vehicle communications in an abstract platform
[TPS_APSD_01002]	Agnosticism of deployment modeling artifacts in an abstract platform
[TPS_APSD_01003]	Exclusion of abstract platform artifacts to an AUTOSAR concrete platform
[TPS_APSD_01004]	System category for a system description with Abstract Platform content
[TPS_APSD_01005]	Identification of component types in an abstract platform
[TPS_APSD_01006]	Recursive component definition in an abstract platform
[TPS_APSD_01007]	Prototyping of ports in an abstract platform
[TPS_APSD_01008]	Generic typing of interfaces in an abstract platform
[TPS_APSD_01009]	Grouping of ports in an abstract platform
[TPS_APSD_01010]	Agnosticism of abstract platform interfaces to middleware deployments
[TPS_APSD_01011]	Aggregation of interface elements in an abstract platform interface
[TPS_APSD_01012]	Modeling of connectors in an abstract platform
[TPS_APSD_01013]	Abstraction of implementation details of data types in an abstract platform
[TPS_APSD_01014]	Allowed data types in an abstract platform
[TPS_APSD_01015]	Deferral of the category of an ApplicationDataType typing in an abstract platform

▽

△

Number	Heading
[TPS_APSD_01016]	Concrete definition of a deferred type
[TPS_APSD_01017]	The category of a deferred type in an abstract platform
[TPS_APSD_01018]	Exclusion of type mapping in an abstract platform
[TPS_APSD_01100]	Requirement annotation in an abstract platform
[TPS_APSD_01101]	Requirements tracing in an abstract platform
[TPS_APSD_01102]	Functional tracing in an abstract platform

Table C.1: Added Traceables in R19-11

C.1.2 Changed Traceables in R19-11

none

C.1.3 Deleted Traceables in R19-11

none

C.1.4 Added Constraints in R19-11

Number	Heading
[constr_6800]	Non-relevance of FibexElement and SystemMapping for a System description with Abstract Platform content
[constr_6801]	Non-relevance of the attributes System.pncVectorLength, System.pncVectorOffset for a System description with Abstract Platform content
[constr_6802]	Restriction of the category of a CompositionSwComponentType which types a RootSwCompositionPrototype in a System description with Abstract Platform content
[constr_6803]	Restriction of the category of a CompositionSwComponentType which references a SwComponentPrototype in a System description with Abstract Platform content
[constr_6804]	Non-relevance of ConstantSpecificationMappingSet and DataTypeMappingSet for a CompositionSwComponentType in an Abstract Platform
[constr_6805]	Non-relevance of PRPortPrototype for a System with Abstract Platform content
[constr_6806]	Restriction of the category of a PortInterface for a System description with Abstract Platform content
[constr_6807]	Exclusivity of an ApplicationInterface to an Abstract Platform
[constr_6808]	Non-relevance of the attribute ClientServerOperation.fireAndForget for a ClientServerOperation used in a ApplicationInterface

▽

△

Number	Heading
[constr_6809]	Non-relevance of ApApplicationError and ApApplicationErrorSet for a ClientServerOperation in the context of a ApplicationInterface
[constr_6810]	Applicable categories for data types in an abstract platform
[constr_6811]	Exclusivity of ApplicationDataType.category DEFERRED to the *abstract platform*
[constr_6812]	SwDataDefProps applicable to ApplicationDataTypes exclusive to the *abstract platform*
[constr_6813]	Restriction of SwComponentTypes in an Abstract Platform

Table C.2: Added Constraints in R19-11

C.1.5 Changed Constraints in R19-11

none

C.1.6 Deleted Constraints in R19-11

none

C.2 Constraint and Specification Item History of this document according to AUTOSAR Release R20-11

C.2.1 Added Traceables in R20-11

Number	Heading
[TPS_APSD_01019]	Typing of SwComponentPrototypes used in a CompositionSwComponentType in an abstract platform
[TPS_APSD_01020]	Semantics of a CompositionSwComponentType of category XP_COMPONENT_APPLICATION
[TPS_APSD_01022]	Semantics of a ApplicationInterface
[TPS_APSD_01023]	Elements of a ApplicationInterface
[TPS_APSD_01024]	Semantics of a ApplicationInterface.command
[TPS_APSD_01025]	Semantics of a ApplicationInterface.indication
[TPS_APSD_01026]	Semantics of a ApplicationInterface of category XP_PORT_CTRL_SECURITY
[TPS_APSD_01027]	Semantics of a ApplicationInterface of category XP_PORT_CTRL_TIMESYNC

▽

△

Number	Heading
[TPS_APSD_01028]	Semantics of a ApplicationInterface of category XP_PORT_DATA_STORAGE
[TPS_APSD_01029]	Semantics of a ApplicationInterface of category XP_PORT_DATA_APPLICATION
[TPS_APSD_01030]	Exclusion of implementation level data types
[TPS_APSD_01031]	Sub-classes of ApplicationDataType
[TPS_APSD_01032]	Semantics of an ApplicationDeferredDataType
[TPS_APSD_01033]	Traceability of an ApplicationDeferredDataType
[TPS_APSD_01034]	Requirement annotation and in an abstract platform
[TPS_APSD_01035]	Placement of an abstract platform model
[TPS_APSD_01036]	Data type tracing between abstract and concrete platform models
[TPS_APSD_01037]	Compatibility of an ApplicationDeferredDataType

Table C.3: Added Traceables in R20-11

C.2.2 Changed Traceables in R20-11

Number	Heading
[TPS_APSD_01000]	Principle of an abstract platform system description
[TPS_APSD_01001]	VFB level modeling of an abstract platform
[TPS_APSD_01002]	Agnosticism of deployment aspects
[TPS_APSD_01003]	Exclusion of abstract platform artifacts to an AUTOSAR concrete platform
[TPS_APSD_01005]	Identification of component types in an abstract platform
[TPS_APSD_01008]	Generic typing of interfaces in an abstract platform
[TPS_APSD_01012]	Modeling of connectors in an abstract platform
[TPS_APSD_01013]	Usage of application level data types
[TPS_APSD_01014]	Allowed data types in an abstract platform
[TPS_APSD_01015]	Deferral of the category of data type in an abstract platform
[TPS_APSD_01016]	Concrete data type resolution of an ApplicationDeferredDataType

Table C.4: Changed Traceables in R20-11

C.2.3 Deleted Traceables in R20-11

Number	Heading
[TPS_APSD_01011]	Aggregation of interface elements in an abstract platform interface
[TPS_APSD_01017]	The category of a deferred type in an abstract platform

▽

△

Number	Heading
[TPS_APSD_01018]	Exclusion of type mapping in an abstract platform
[TPS_APSD_01100]	Requirement annotation in an abstract platform
[TPS_APSD_01101]	Requirements tracing in an abstract platform
[TPS_APSD_01102]	Functional tracing in an abstract platform

Table C.5: Deleted Traceables in R20-11

C.2.4 Added Constraints in R20-11

Number	Heading
[constr_6814]	Restriction of ApplicationDeferredDataType.category

Table C.6: Added Constraints in R20-11

C.2.5 Changed Constraints in R20-11

Number	Heading
[constr_6803]	Standarized values of CompositionSwComponentType.category
[constr_6806]	Standarized values of ApplicationInterface.category

Table C.7: Changed Constraints in R20-11

C.2.6 Deleted Constraints in R20-11

Number	Heading
[constr_6800]	Non-relevance of FibexElement and SystemMapping for a System description with Abstract Platform content
[constr_6801]	Non-relevance of the attributes System.pncVectorLength, System.pncVectorOffset for a System description with Abstract Platform content
[constr_6802]	Restriction of the category of a CompositionSwComponentType which types a RootSwCompositionPrototype in a System description with Abstract Platform content
[constr_6804]	Non-relevance of ConstantSpecificationMappingSet and DataTypeMappingSet for a CompositionSwComponentType in an Abstract Platform
[constr_6805]	Non-relevance of PRPortPrototype for a System with Abstract Platform content
[constr_6808]	Non-relevance of the attribute ClientServerOperation.fireAndForget for a ClientServerOperation used in a ApplicationInterface
[constr_6809]	Non-relevance of ApApplicationError and ApApplicationErrorSet for a ClientServerOperation in the context of a ApplicationInterface

▽

△

Number	Heading
[constr_6811]	Exclusivity of ApplicationDataType.category DEFERRED to the *abstract platform*
[constr_6813]	Restriction of SwComponentTypes in an Abstract Platform

Table C.8: Deleted Constraints in R20-11

C.3 Constraint and Specification Item History of this document according to AUTOSAR Release R21-11

C.3.1 Added Traceables in R21-11

Number	Heading
[TPS_APSD_01038]	Semantics of a ApplicationInterface of category XP_PORT_SAFETY
[TPS_APSD_01039]	Semantics of a ApplicationInterface.attribute
[TPS_APSD_01040]	Attributes of a Field

Table C.9: Added Traceables in R21-11

C.3.2 Changed Traceables in R21-11

Number	Heading
[TPS_APSD_01023]	Elements of a ApplicationInterface
[TPS_APSD_01026]	Semantics of a ApplicationInterface of category XP_PORT_SECURITY
[TPS_APSD_01027]	Semantics of a ApplicationInterface of category XP_PORT_TIMESYNC
[TPS_APSD_01028]	Semantics of a ApplicationInterface of category XP_PORT_STORAGE
[TPS_APSD_01029]	Semantics of a ApplicationInterface of category XP_PORT_APPLICATION

Table C.10: Changed Traceables in R21-11

C.3.3 Deleted Traceables in R21-11

Number	Heading
[TPS_APSD_01013]	Usage of application level data types

Table C.11: Deleted Traceables in R21-11

C.3.4 Added Constraints in R21-11

none

C.3.5 Changed Constraints in R21-11

Number	Heading
[constr_6806]	Standardized values of ApplicationInterface.category

Table C.12: Changed Constraints in R21-11

C.3.6 Deleted Constraints in R21-11

none

D Splitable Elements in the Scope of this Document

This chapter contains a table of all model elements stereotyped ≪atpSplitable≫ in the scope of this document.

Each entry in the table consists of the identification of the specific model element itself and the applicable value of the tagged value atp.Splitkey.

For more information about the concept of splitable model elements and how these shall be treated please refer to [5].

E Variation Points in the Scope of this Document

This chapter contains a table of all model elements stereotyped ≪atpVariation≫ in the scope of this document.

Each entry in the table consists of the identification of the model element itself and the applicable value of the tagged value vh.latestBindingTime.

For more information about the concept of variation points and how model elements that contain variation points shall be treated please refer to [5].

Variation Point	***Latest Binding Time***
ApplicationInterface.attribute	blueprintDerivationTime
ApplicationInterface.command	blueprintDerivationTime
ApplicationInterface.indication	blueprintDerivationTime

Table E.1: Usage of variation points