Document Title	Integration of Franca IDL Software Component Descriptions
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	663

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R21-11

Document Change History

Date	Release	Changed by	Description
2021-11-25	R21-11	AUTOSAR Release Management	• No content changes
2020-11-30	R20-11	AUTOSAR Release Management	• No content changes
2019-11-28	R19-11	AUTOSAR Release Management	• Editorial changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Editorial changes
2015-07-31	4.2.2	AUTOSAR Release Management	• Editorial changes
2014-10-31	4.2.1	AUTOSAR Release Management	• Initial Release

Disclaimer

Table of Contents

References

[1] Franca User Guide
https://code.google.com/a/eclipselabs.org/p/franca/downloads/detail?name=FrancaUserGuide-0.3.0.pdf

[2] Virtual Functional Bus
AUTOSAR_EXP_VFB

[3] Specification of RTE Software
AUTOSAR_SWS_RTE

[4] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[5] Methodology for Classic Platform
AUTOSAR_TR_Methodology

[6] IPC CommonAPI C++
http://projects.genivi.org/commonapi/

[7] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes

1 Introduction

1.1 Objective

AUTOSAR covers different automotive application domains, but not necessarily all of them. Instead of trying to ever extend AUTOSAR to make it easily applicable to domains that are yet difficult to implement in AUTOSAR, it seems more reasonable to open AUTOSAR for an integration with standards and technologies that are specifically designed fur such application domains. The open-source development platform GENIVI (see www.genivi.org) for instance defines a standard and technology for in-vehicle infotainment systems and is supported and employed by many companies. The GENIVI architecture is similar to the one of AUTOSAR in that it distinguishes application level, middleware, and basic software, which facilitates the integration. For the description of the software components at the application level GENIVI uses the Franca Interface Definition Language (Franca IDL, see [1]).

Also the processes of AUTOSAR and GENIVI are similar. Both strive for a generation of middleware and basic software from a description of the application level components and their distribution onto a network of ECUs. Therefore it is possible also to split the integration of AUTOSAR and GENIVI systems into an application level part and a communication level part.

The purpose of the Franca Integration is to support the integration of AUTOSAR and GENIVI systems at the application level. That means that a virtual integration of functions is addressed, corresponding to the Virtual Functional Bus view of AUTOSAR (see [2]). The Franca Integration provides a notation for the specification of the connections of the AUTOSAR and the GENIVI application components and a bidirectional translation between the descriptions of these components. With these means the Franca Integration makes it possible to interconnect the development and generation processes of the AUTOSAR and the GENIVI parts of the overall system.

This application level integration has to be combined with a communication level integration that realizes the message exchange among the AUTOSAR and the GENIVI systems *on the wire*. That means that common protocols and means for the generation of basic software and middleware from software and system descriptions have to be provided. This level is addressed in other AUTOSAR contributions, for instance by the serialization protocol SOME/IP for the communication via Ethernet.

1.2 Goal

When an AUTOSAR system and a GENIVI system are developed, their application level components are described using the formats defined by the two standards: an AUTOSAR software component description for the AUTOSAR part and a Franca IDL description for the GENIVI part. The AUTOSAR software component description is given by one or several arxml-files that contain the XML-representation of the description; the Franca description is given by one or several fidl- and fdepl-files that contain

the textual representation of the description according to the textual grammar defined by the Franca IDL. In this process state there is no complete description of the integrated system in either format yet, nor can the desired inter-operation of the two systems be described in one of the formats. This is due to the fact that the names of the methods, operations, attributes, etc. of the respective other part are not yet contained in the description of the own part. These two features – (1) a description of the interconnections for the inter-operation and (2) a complete system description – shall be achieved by the Franca Integration. For that purpose it comprises three parts.

1. A new format for the specification of the application level interconnection of the AUTOSAR and the GENIVI part, the *Franca Connector*.
2. A translation of Franca models with Franca Connectors to AUTOSAR software component descriptions.
3. A translation of AUTOSAR software component descriptions to Franca models.

The Franca Connector shall be used to specify which GENIVI component calls which AUTOSAR component and vice versa. Although the Franca IDL contains an extension mechanism – the deployment specification – that would allow the definition of the desired interconnection within Franca IDL, the Franca Connector is defined as a new format. The reason for that is to support an easy generalization of the integration approach to other component or interface description languages. Moreover, this approach also leaves open the possibility to define the desired interconnection by a Franca deployment definition, and then to generate the corresponding Franca Connector from this deployment definition, or to generate the Franca deployment definition from the Franca Connector.

Given a specification of the desired interconnection by a Franca Connector, the two translations make it possible to obtain a description of the complete, integrated system at the application level in either format: an AUTOSAR software component description or a Franca model. It is important to note, however, that a Franca model only addresses the type level – component types and data types – whereas an AUTOSAR description in addition specifies component instances (called prototypes) and their connections. Moreover, the AUTOSAR data types are much more detailed than the corresponding data type definitions in Franca. For these reasons the complete integrated application level Franca model and the complete integrated application level AUTOSAR description will not be semantically equivalent. They will be consistent, but both the scope and the detailing of the AUTOSAR description are larger.

Seen from the AUTOSAR perspective we state the achievement of the complete system descriptions as the overall goal of the Franca Integration:

[TR_FRANCA_00000] Goal of the Franca Integration ⌈The goal of the Franca Integration is to obtain two consistent complete descriptions of the application level of a system that consists of AUTOSAR parts and parts that are described with the Franca IDL: one as an AUTOSAR software component description and one as a Franca model. The completeness of the descriptions is thereby relative to the expression means of the concerned description format.⌋ *()*

1.3 Motivation

In order to motivate in more detail the need for the Franca Integration as part of an integration of an AUTOSAR and a non-AUTOSAR system, we sketch as an overall use case the development of an integrated system in which an automotive application component *AutoComp* and an infotainment application component *InfoComp* inter-operate (see Figure 1.1). Thereby the former is a part of an AUTOSAR system and the latter is a part of a GENIVI system. The integrated system might include e.g. the following two inter-operations.

- *InfoComp* requests a service from *AutoComp*, e.g. information on a vehicle state.
- *AutoComp* requests a service from *InfoComp*, e.g. information for diagnosis.

For sake of simplicity we assume that the two components run on different ECUs, an AUTOSAR ECU and a GENIVI ECU, that are connected via Ethernet. The reason for assuming this setting is that with SOME/IP there is already a fitting protocol that can be implemented both within AUTOSAR and GENIVI. Other system configurations, e.g. a connection via a slow bus like SPI or a solution where the two systems run on the same processor, would require other protocols. In the considered setting the basic side conditions of the overall AUTOSAR-GENIVI integration can be formulated as follows.

- *AutoComp* is realized as an AUTOSAR software component for an AUTOSAR ECU, that means
  1. *AutoComp* uses for the communication only the AUTOSAR communication services provided via the software component API of the AUTOSAR Run Time Environment (RTE, see [3]).
  2. *AutoComp* has an AUTOSAR software component description (see [4]).
  3. RTE and basic software of the AUTOSAR ECU are generated and configured according to the AUTOSAR process (see [5]).
- *InfoComp* is realized as a GENIVI component for a GENIVI ECU, that means
  1. *InfoComp* uses for the communication only the services provided by GENIVI Inter Node Communication Middleware (INC MW) and Transport Protocol (INC TP) via the Common API (see [6]).
  2. *InfoComp* has a description of the interfaces it implements in Franca IDL.
  3. The implementation of *InfoComp* depends only on the Common API stubs and proxies generated from Franca IDL descriptions.

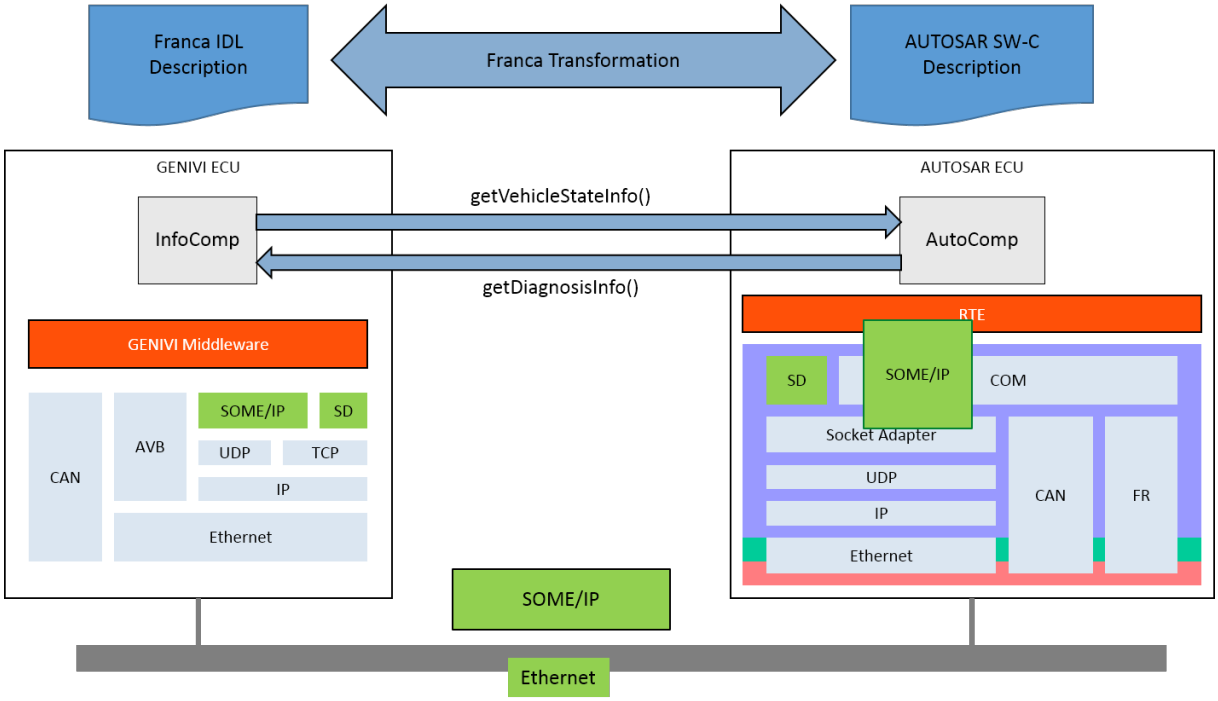

Figure 1.1: Inter-operation of GENIVI and AUTOSAR Application Components

According to these conditions, in order to be able to communicate with *InfoComp* the AUTOSAR component *AutoComp* first needs an RTE API operation to call the desired *InfoComp* method *getDiagnosisInfo()* at one of its ports. Secondly, it needs an Ethernet communication stack that realizes the signal routing to the bus. RTE API and communication stack are only generated properly if the communication link between *AutoComp* and *InfoComp* is contained in the AUTOSAR software component description as a connector. This in turn is only possible if there is also a representation of *InfoComp* in the AUTOSAR Software Component Description. In order to obtain that, the translation of Franca Models to AUTOSAR software component descriptions is needed.

The same holds for the generation of the Common API for the GENIVI part of the system: it needs information on the AUTOSAR components it wants to communicate with, specified in Franca IDL. Given the translation from AUTOSAR software component descriptions to Franca models this can also be achieved.

1.4 Integration Method

The AUTOSAR process as described in [5] starts with a description of the system with the AUTOSAR notation. That means that the corresponding templates for the description of the application components and their connections, the ECUs and their connections, and the mapping of the application components to the ECUs are filled. The Franca Integration addresses a methodological step that lies ahead of this starting point. When it starts only incomplete descriptions are provided – in particular at the

application level – because the interconnection of the AUTOSAR and GENIVI application components is not yet specified. The description of the integrated system is only the goal of the Franca Integration.

The initial situation of a Franca Integration can be defined as follows.

- There is an AUTOSAR software component description of application components that are connected among each other. Some ports may not be connected and some ports may have no or incomplete interfaces. These represent either operations offered to the GENIVI part (provided port not connected) or operations required from the GENIVI part (no or incomplete required port interface).
- There is a Franca model that contains a set of interface and data type definitions.
- It is known – but not yet formally represented – which AUTOSAR component shall inter-operate with which GENIVI component and vice versa. Inter-operation may consist of a client server communication or a sender receiver communication.

The main methodical steps of the Franca Integration in this situation as seen from the AUTOSAR perspective are:

1. Represent the knowledge on the inter-operation by a Franca Connector.
2. Apply the Franca-to-AUTOSAR translation to the Franca model and the Franca Connector.

The result is an AUTOSAR software component description of the complete, integrated application level of the system, i.e. a complete VFB view.

The GENIVI perspective is analogous. Due to the fact that instances and connections are not represented in Franca IDL the Franca Connector is not relevant for the derivation of the complete Franca Model. Thus there is only one step.

1. Apply the AUTOSAR-to-Franca translation to the AUTOSAR software component description.

The result is a Franca model of the complete, integrated application level of the system. It consists of the complete set of interfaces of the system, the ones of the GENIVI part and the ones of the AUTOSAR part.

As mentioned above, the two complete integrated application level descriptions are semantically consistent, but not equivalent. First of all this is due to the fact that a Franca model specifies types, but no instances or connections. Moreover, a Franca interface defines only the methods and attributes a component offers (provides), not the ones it requires. In Figure 1.2 the different aspects addressed by Franca models and AUTOSAR software component descriptions are depicted. Both specify data types and interfaces. Component instances and intra-connections (i.e. connections among component instances within either the GENIVI or the AUTOSAR part of the system) are only specified in the AUTOSAR software component description. Interconnections (i.e. connections between an AUTOSAR and a GENIVI component instance) are obviously specified neither in a Franca model nor in an AUTOSAR software component description. This dissymmetry is captured by the Franca Connector. It offers the possibility to

define component instances that implement Franca interfaces and interconnections of Franca and AUTOSAR component instances. This is defined in detail in chapter 2.

As a consequence, also the two translations have different results. The Franca-to-AUTOSAR translation takes the information from the Franca model and the Franca Connector and constructs an AUTOSAR software component description that contains the Franca interfaces, component instances, and interconnections as port interfaces, component prototypes, and connectors respectively. The AUTOSAR-to-Franca translation only considers the port interfaces and data types of the AUTOSAR software component description and translates them to the Franca IDL. The instances and interconnections cannot be represented in Franca IDL, anyway.

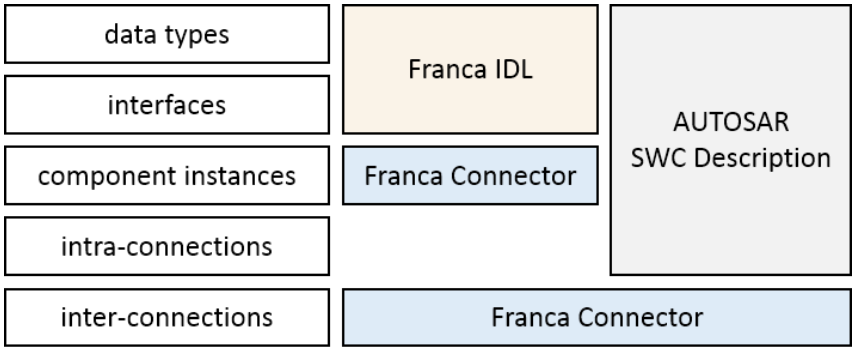

Figure 1.2: Scopes of Franca Models and AUTOSAR Software Component Descriptions

1.4.1 Integrated System Description as AUTOSAR SWC Description

Figure 1.3 shows an example for the Franca-to-AUTOSAR translation in a scenario where an AUTOSAR operation requests a GENIVI method. Initially the following specifications are given (depicted in black in Figure 1.3).

- The Franca model defines an interface *F* that contains a method *m*.
- The AUTOSAR software component description defines a component type *A* and an instance *a* of *A* in a composition type *AC*.
- *A* has a required port *p* where the Franca method *m* shall be called. The interface of this port is not yet defined since there is no representation within the AUTOSAR software component description of *m*.
- The Franca Connector specifies
  - that there is a component instance *f* that implements the interface *F* and
  - that the required port *p* of the AUTOSAR component instance *a* is connected with the interface *F* provided by the Franca instance *f*.

The Franca-to-AUTOSAR translation then adds the following parts to the AUTOSAR software component description (depicted in blue in Figure 1.3).

- An interface that contains an operation *m* and a declaration that the required AUTOSAR port *p* is typed by this interface.
- A component type *F* with a provided port that is also typed by this interface.
- An instance *f* of *F* in the composition type *AC*.
- A connector of the open AUTOSAR port *p* and the port of the new component type *F* in the composition type *AC*.

Thus within the AUTOSAR software component description now the desired interconnection of the AUTOSAR component instance *a* and the Franca component instance *f* is represented.

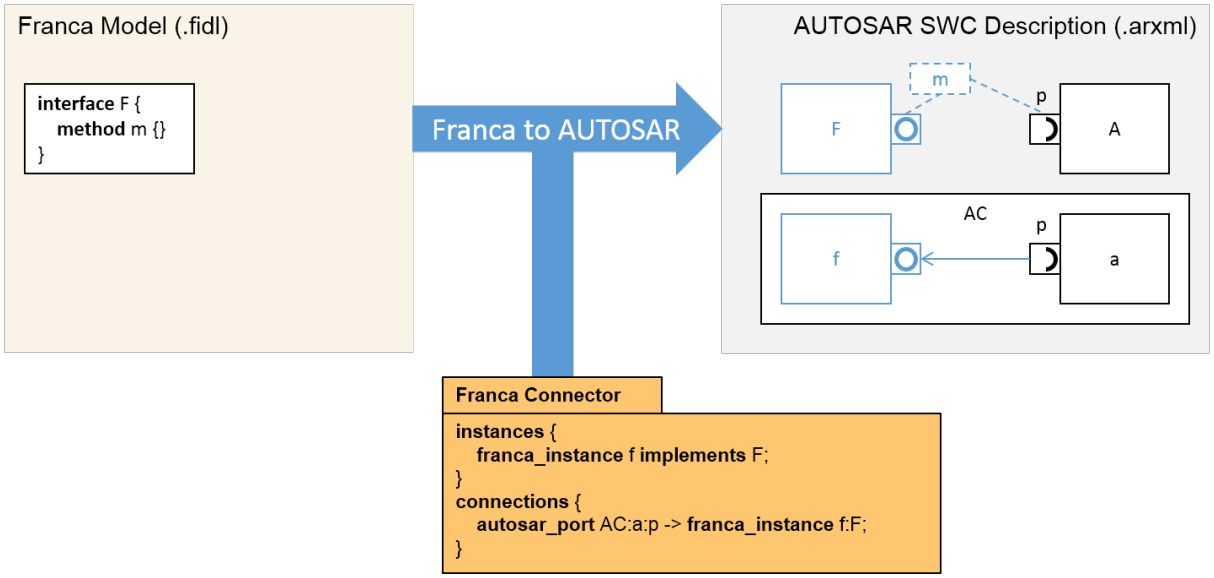

Figure 1.3: Franca-to-AUTOSAR Translation

The opposite scenario – a GENIVI methods requests an AUTOSAR operation – is of no further interest for the Franca-to-AUTOSAR translation, because a request is not represented in a Franca interface. The Franca interface could be translated to an AUTOSAR component type, but neither a new AUTOSAR instance would be generated nor a connection.

Sender receiver communication instead of client server communication (operation calling) is handled in the same way as the operation call scenario described above. The provision of signals is expressed in Franca IDL by broadcasts. A Franca instance implementing an interface that contains a broadcast is translated to an AUTOSAR component that offers a data element of the same data type as the broadcast at a provided port. The latter can be connected to a port of an AUTOSAR component that requires the data element.

1.4.2 Integrated System Description as Franca Model

A scenario in which an AUTOSAR component offers an operation for a GENIVI component is depicted in Figure 1.4. In this case the AUTOSAR Software Component Description is complete, but there is no component instance that requests the operation *op* provided at port *q* of component *B*. The Franca model is yet empty because the request for an operation cannot be expressed. The information that there is an instance *g* that requests the AUTOSAR operation *op* is represented in the Franca Connector. In this (artificial) example *g* is a Franca component instance that implements none of the considered Franca interfaces. It is only introduced to define within the complete system description who calls the operation at port *q* of the AUTOSAR component instance *b*.

The AUTOSAR-to-Franca translation adds an interface *B* with a method *op* to the Franca Model that can now be used by other Franca components. Since instances and connections are not represented in Franca IDL this is all the translation does.

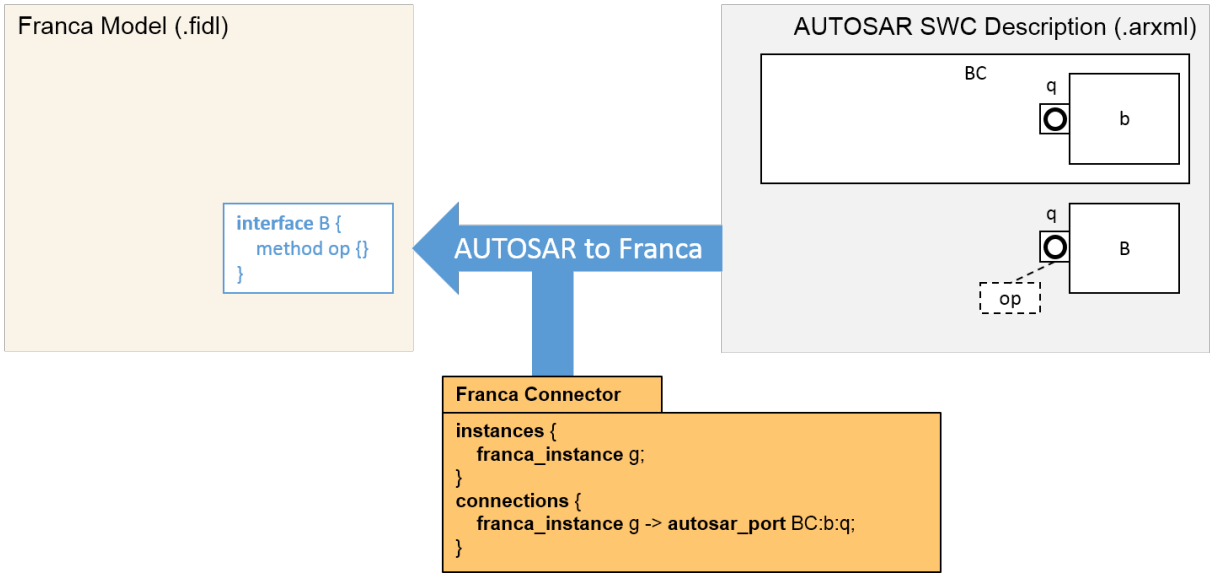

Figure 1.4: AUTOSAR-to-Franca Translation

1.4.3 Complete View

Putting together the scenarios discussed above we obtain the two complete application level system views that have been announced as goal of the Franca Integration. Figure 1.5 shows the initial situation: descriptions of application components as Franca models, descriptions of application components as AUTOSAR software component descriptions, and a Franca Connector. The results of the Franca-to-AUTOSAR translation and the AUTOSAR-to-Franca translation are shown in Figure 1.6. The AUTOSAR description is extended by component types (`AtomicSwComponentType`s) and instances (`SwComponentPrototype`) for the Franca interfaces and instances, an interface that contains the method that is offered by a Franca component and requested

by an AUTOSAR component, and the two connections that correspond to the two connection entries in the Franca Connector. The Franca Model is extended by interface definitions for the AUTOSAR component types.

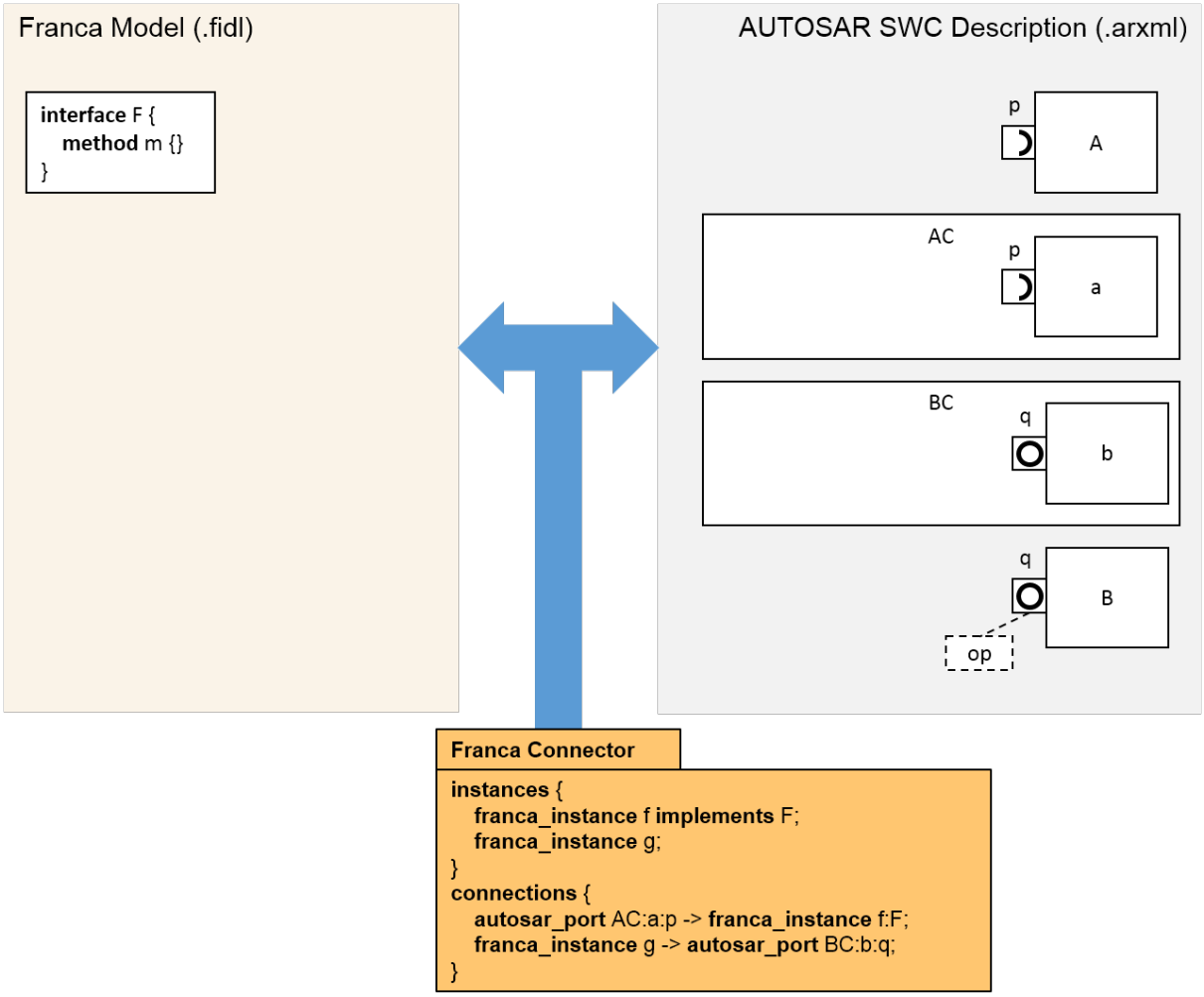

Figure 1.5: Initial State of the Franca Integration

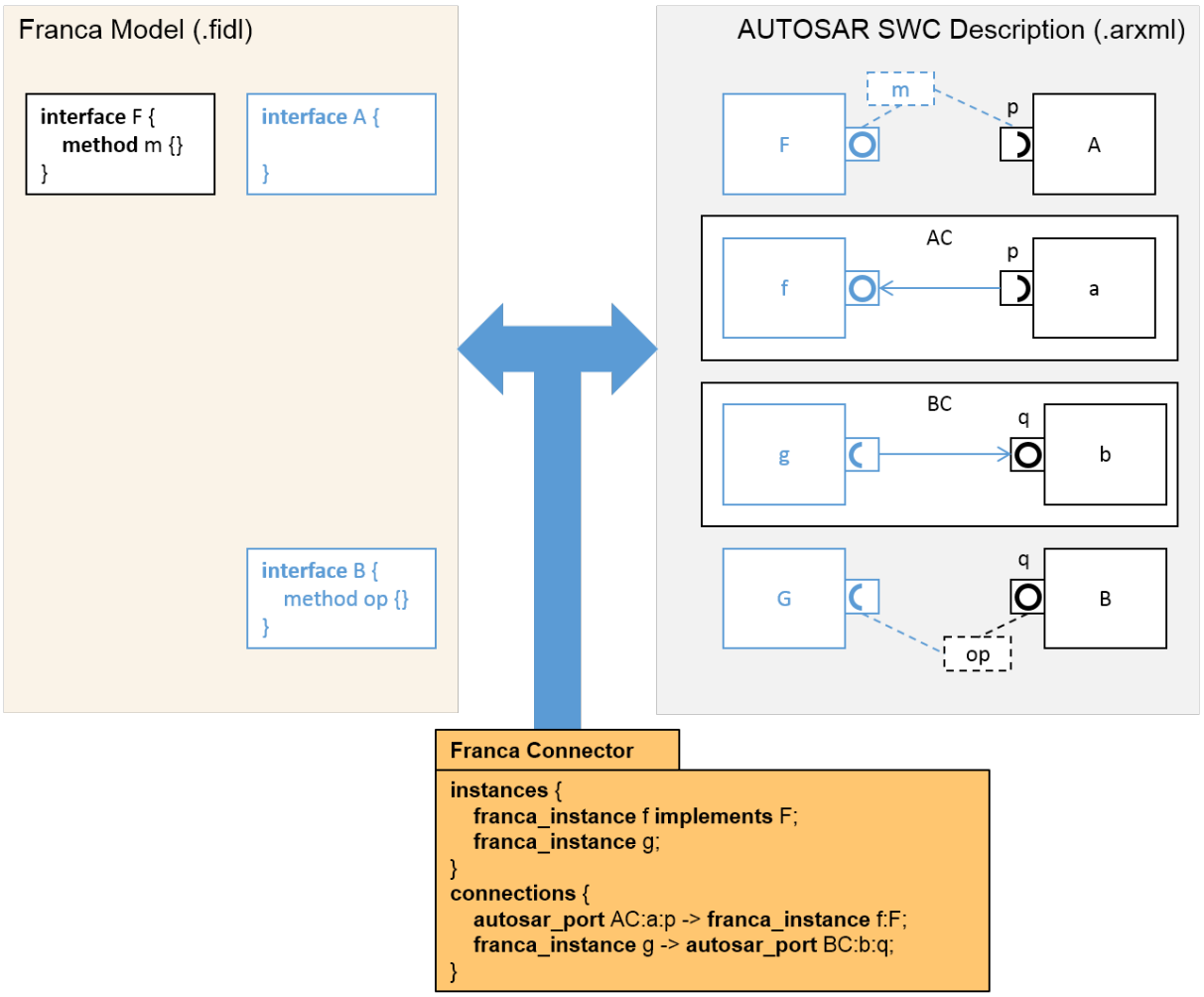

Figure 1.6: Integrated System Views in Franca and AUTOSAR

1.5 Limitations and Extensions

1.5.1 Dynamic Communication

The AUTOSAR process requires that all inter-operations among application component instances that may occur during run time are declared statically (before compile time) in the AUTOSAR software component description. Inter-operation in an infotainment system on the other hand is typically dynamic. GENIVI for instance uses sockets that allow the dynamic discovery and connection with service providers at run time. Future AUTOSAR releases may support dynamic communication, too, but in the present state static declaration of communication links is mandatory. Thus at least the AUTOSAR and the GENIVI component instances that shall inter-operate must be known and identified at design time. At the GENIVI side it may be possible to introduce these component instances as place holders and establish the connection with the real component instances at run time via a corresponding dynamic discovery and connection

service. At the AUTOSAR side the instances have to be declared at design time anyway, so they are present and can be used for the specification of the interconnection. With a solution of this kind the static interconnection declaration would be limited to the AUTOSAR part that underlies this restriction anyway, whereas the GENIVI part would not be constrained. A more detailed discussion of the integration of dynamic communication with an AUTOSAR system is necessary, but not in the scope of this report.

1.5.2 RTE Contract and RTE Generation

The Franca Integration aims at a Virtual Functional Bus View of the integrated system, which is only the first step of an AUTOSAR development. In order to generate the AUTOSAR RTE further information on the ECU network, the application components, and the mapping of the application components to the ECU network is needed. This information is defined in [3]. In the first step, the RTE Contract Phase, the behavior of the components needs to be defined and implemented and the information on the data types has to be refined. The second one, the RTE Generation Phase, also requires information on the ECU resources and the mapping of the application components to the resources. For a complete integration of an AUTOSAR and a GENIVI system these phases and the corresponding description requirements have to be considered, too. Since the Franca IDL has no fixed means to specify behavior, resources, or allocations the Franca Integration cannot define corresponding translations. It would rather be a task to define a Franca deployment specification for the AUTOSAR integration that covers these aspects.

2 Franca Connector

The Franca connector is the new format that is introduced to specify the desired inter-operation of the Franca and the AUTOSAR application components. It consists of three major parts:

Imports References to the Franca models and the AUTOSAR software component descriptions that define the Franca and the AUTOSAR application components respectively.

Franca Instances Definitions of the Franca component instances that shall take part in the desired inter-operations.

Links Definitions of the interconnections of AUTOSAR and Franca component instances.

2.1 Imports and Franca Instances

An **import** is a string that indicates the location of a Franca model (fidl-file) or an AUTOSAR software component description (arxml-file). The imports define in particular the Franca interfaces and AUTOSAR ports that can be referenced in the Franca Connector.

A **Franca instance** is declared by its name and the list of Franca interfaces it implements. The Franca interfaces must be contained in the imported Franca models. The list of implemented interfaces of an instance may be empty.

A possible concrete notation for a Franca instance definition in a Franca Connector is

franca_instance g **implements** $F1, \ldots, Fn$

where g is the name of the defined Franca instance and $F1, \ldots, Fn$ are the names of the implemented Franca interfaces.

2.2 Links

A **link** has an AUTOSAR side and a Franca side. The AUTOSAR side is always given by a port instance reference, i.e. a `SwComponentPrototype` and a `PortPrototype` that belongs to the `SwComponentType` of the `SwComponentPrototype`. A possible concrete notation for the AUTOSAR side is ***autosar_port*** *comp : p* where *comp* is the name of the `SwComponentPrototype` and *p* is the name of the `PortPrototype`.

The Franca side of a link is given either by a Franca instance alone or by a Franca instance and one of the Franca interfaces it implements.

franca_instance *g:F* or ***franca_instance*** *g*

where *g* is the name of the Franca instance and *F* is the name of the Franca interface.

A link is directed in the sense of the intended communication flow. The left side of the link defines the instance that emits the data element or the operation call; the right side defines the instance that receives the data element or operation call.

Each AUTOSAR port is typed by an interface which may be a client server interface or a sender receiver interface. In the first case it contains operations that are either offered (provided, `PPortPrototype`) or required (`RPortPrototype`) at the port. In the second case it contains data elements that are either sent (provided, `PPortPrototype`) or expected (required, `RPortPrototype`) at the port. The two kinds of AUTOSAR interfaces and two directions of Franca Connector links (AUTOSAR-to-Franca and Franca-to-AUTOSAR) yield four types of links.

1. AUTOSAR-to-Franca Client Server Link
2. AUTOSAR-to-Franca Sender Receiver Link
3. Franca-to-AUTOSAR Client Server Link
4. Franca-to-AUTOSAR Sender Receiver Link

Figure 2.1 shows examples for the four possible types of links. It uses a mixed notation and is only intended to explain the Franca link types, their representation within a Franca connector, and the result of the translation to AUTOSAR. Components and ports are shown in an AUTOSAR style. The links are depicted, for sake of brevity, at the type level; in a more faithful figure the connections of the port instance references should be shown. The connectors are labeled using the concrete notation for links introduced above, where for sake of brevity the labels **autosar_port** and **franca_instance** are omitted. The names of the links (*AF_CS*, *AF_SR*, *FA_CS*, and *FA_SR*) indicate the direction (AUTOSAR-to-Franca or Franca-to-AUTOSAR) and the AUTOSAR port interface type (Client-Server or Sender-Receiver). The distinction of the two Autosar-to-Franca sender receiver links *AF_SRPull* and *AF_SRPull* is due to the possibility in Franca to declare methods as fire-and-forget methods. This will be explained in more detail below and in subsection 3.4.2.

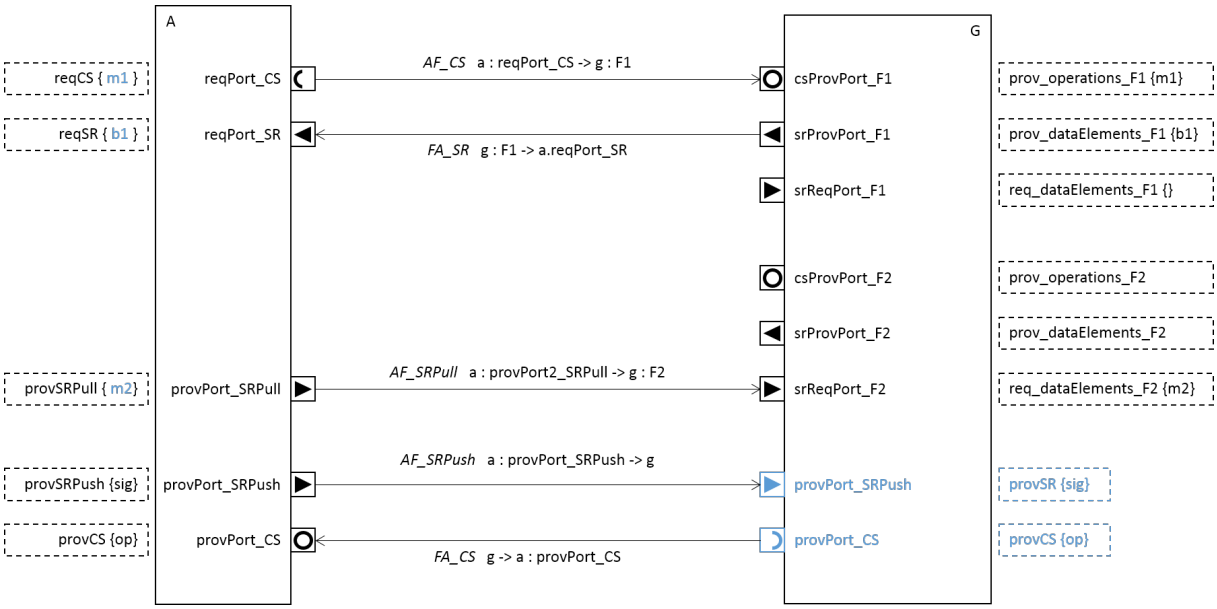

Figure 2.1: Links of AUTOSAR and Franca component instances

In the discussion below we assume that the following AUTOSAR and Franca elements are given as starting point.

1. An AUTOSAR component *A* with ports as defined in Table 2.1.
2. An AUTOSAR component prototype *a* whose type is *A*.
3. A Franca Interface *F1* with a method *m1* and a broadcast *b1*, and a second Franca Interface *F2* with a fire-and-forget method *m2*.
4. A Franca instance *g* that implements *F1* and *F2*.

port	interface	interface contents
reqPort_CS	*reqCS*	∅
reqPort_SR	*reqSR*	∅
provPort_CS	*provCS*	{ *op* }
provPort_SRPush	*provSRPush*	{ *sig* }
provPort_SRPull	*provSRPull*	∅

Table 2.1: Ports of AUTOSAR component *A*

The translation of the Franca interfaces and the Franca instance to AUTOSAR – that is discussed in the following chapter – yields the component type shown on the right side of Figure 2.1. For each Franca interface (for example *F1*) there three three ports,

1. one that provides the methods of the Franca interface as AUTOSAR operations (*csProvPort_F1* typed by *prov_operations_F1*).
2. one that provides the broadcasts of the Franca interface as AUTOSAR data elements (*srProvPort_F1* typed by *prov_dataElements_F1*).
3. one that requests the fire-and-forget methods of the Franca interface as AUTOSAR data elements (*srReqPort_F1* typed by *req_dataElements_F1*).

The five connectors are generated by the five Franca links as discussed next.

2.2.1 AUTOSAR-to-Franca Client Server Link

An AUTOSAR-to-Franca client server link

autosar_port *a : reqPort_CS* → **franca_instance** *g : F1*

specifies that the AUTOSAR component prototype *a* requires (calls) at its port *reqPort_CS* the operations (methods) defined in the Franca interface *F1* from the Franca instance *g*. The correctness condition for an AUTOSAR-to-Franca client server link is that the AUTOSAR side of the link is a required port (`RPortPrototype`) typed by a client server interface (`ClientServerInterface`) and that the Franca side has a Franca interface.

2.2.2 AUTOSAR-to-Franca Sender Receiver Link

There are two kinds of AUTOSAR-to-Franca sender receiver links that are distinguished by their Franca sides. If the Franca side contains an interface it means that the Franca instance that implements this interface offers a fire-and-forget method. The link

autosar_port *a : provPort_SRPull* → **franca_instance** *g : F2*

states that the fire-and-forget method is called by the AUTOSAR component prototype *a*. The fire-and-forget method that has not been known in the AUTOSAR description yet is *pulled* via the link into the interface that types the AUTOSAR port *provPort_SRPull*. (This is indicated by the blue *m2* in the interface *provSRPull*.)

If the Franca side does not contain an interface the link

autosar_port *a : provPort_SRPush* → **franca_instance** *g*

specifies that the AUTOSAR component prototype *a* sends the data elements declared in the interface *provSRPush* that types the port *provPort_SRPush* to the Franca Instance *g*. Since the Franca model does not specify which data elements can be sent to an instance the corresponding elements are now created. The port *provPort_SRPush*, the interface *provSRPush*, and the data element *sig* provided at port *provPort_SRPush* are *pushed* to the Franca side.

The correctness condition for an AUTOSAR-to-Franca Sender Receiver Link is that the AUTOSAR side is a provided port (`PPortPrototype`) typed by a sender receiver interface (`SenderReceiverInterface`) and that the Franca side either has a Franca interface that contains at least one fire-and-forget method (pull link), or the Franca side has no interface (push link).

2.2.3 Franca-to-AUTOSAR Client Server Link

A Franca-to-AUTOSAR client server link

franca_instance *g* → **autosar_port** *a : provPort_CS*

specifies that the Franca instance *g* requires (calls) AUTOSAR operations. The correctness condition for a Franca-to-AUTOSAR client server link is that the Franca side does not have a Franca interface and that the AUTOSAR side is a provided port (PPortPrototype) typed by a client server interface (ClientServerInterface).

2.2.4 Franca-to-AUTOSAR Sender Receiver Link

A Franca-to-AUTOSAR sender receiver link

franca_instance *g : F1* → **autosar_port** *a : reqPort_SR*

specifies that the Franca instance *g* sends the broadcasts (and the notifications of the attributes) of the Franca interface *F1* to the AUTOSAR port *reqPort_SR*. The correctness condition for a Franca-to-AUTOSAR sender receiver link is that that Franca side must have a Franca interface and the AUTOSAR side is a required port (RPortPrototype) typed by a sender receiver interface (SenderReceiverInterface).

2.3 Constraints

The following constraints must be respected by the set of links contained in a Franca connector.

The first constraint is a formal one; it prevents duplicate links.

[TR_FRANCA_CONSTR_00010] Franca connector has no duplicate links ⌈There must not be two links with the same AUTOSAR and Franca sides in a Franca connector.⌋*()*

The second constraint prevents that a client is connected to more than one server.

[TR_FRANCA_CONSTR_00020] Franca connector has no client server fan out ⌈A required client server port of an AUTOSAR component prototype must not be connected to more than one Franca instance.⌋*()*

3 Franca-to-AUTOSAR Translation

The input for a translation in either direction – Franca to AUTOSAR or AUTOSAR to Franca – is always a Franca Connector. Via its imports the Franca Connector references the Franca models and AUTOSAR software component descriptions that shall be interconnected and translated. The target of a translation can be either an AUTOSAR software component description (Franca-to-AUTOSAR translation) or a Franca model (AUTOSAR-to-Franca translation).

It is possible to define a Franca Connector that consists only of a Franca import; that means that its AUTOSAR import is empty and it does not contain links. In this case the Franca-to-AUTOSAR translation only translates a specification of interfaces and data types in Franca IDL to a semantically equivalent representation of these interfaces and data types as an AUTOSAR XML document.

The more general case is the one in which both Franca and AUTOSAR specifications are imported and the two are connected. In this case the Franca-to-AUTOSAR translation yields an AUTOSAR software component description that contains

- the imported AUTOSAR software component description,
- the translation of the Franca model (interfaces and data types),
- a representation of the interconnections of the Franca and AUTOSAR instances.

A pure translation is thus a special case of the more general integration of Franca models and AUTOSAR software component descriptions

3.1 Notation

The definition of the translation of Franca IDL elements to AUTOSAR elements follows their presentation in [1]. For each Franca IDL metaclass we name a generic element and define the AUTOSAR element or set of elements that this element is mapped to. For that purpose we use a table – or a set of tables, in case the France IDL element is mapped to a set of AUTOSAR elements – with the following meaning.

AR Element	This entry defines the AUTOSAR metaclass the Franca metaclass is mapped to. Moreover, a *name* for the target element is introduced in order to refer to the result of the mapping in further entries or rules.
AR Container	This entry specifies the AUTOSAR element that contains the target element defined in the entry above by its *name*.
Attributes	This entry defines the attributes and cross references of the target element.
Condition	In this entry a condition for the mapping can be given. If the condition is false the Franca element does not generate a target element in the AUTOSAR representation.

3.2 Franca Models

The translation of the top level element `FModel` of a Franca Model yields a structure of AUTOSAR packages that are used later on as containers for the further elements. A top package (the *FrancaModelPackage*) is generated that contains the complete result of the translation. It is added to the root of the AUTOSAR XML.

[TR_FRANCA_00010] Franca model is mapped to AUTOSAR top level package structure ⌈An `FModel` *fModel* is mapped to the set of `ARPackage`s described in Table 3.1, Table 3.2, Table 3.3, Table 3.4, Table 3.5, Table 3.6 and Table 3.7.⌋ *()*

AR Element	`ARPackage` *FrancaModelPackage*
AR Container	*AUTOSAR*
Attributes	shortName = *fModel*.name
Condition	—

Table 3.1: FrancaModelPackage mapping

AR Element	`ARPackage` *FrancaApplicationDataTypes*
AR Container	*FrancaModelPackage*
Attributes	shortName = "FrancaApplicationDataTypes"
Condition	—

Table 3.2: FrancaApplicationDataTypes mapping

AR Element	`ARPackage` *FrancaImplementationDataTypes*
AR Container	*FrancaModelPackage*
Attributes	shortName = "FrancaImplementationDataTypes"
Condition	—

Table 3.3: FrancaImplementationDataTypes mapping

AR Element	`ARPackage` *FrancaBaseDataTypes*
AR Container	*FrancaModelPackage*
Attributes	shortName = "FrancaBaseDataTypes"
Condition	—

Table 3.4: FrancaBaseDataTypes mapping

AR Element	`ARPackage` *FrancaDataTypeMaps*
AR Container	*FrancaModelPackage*
Attributes	shortName = "FrancaDataTypeMaps"
Condition	—

Table 3.5: FrancaDataTypeMaps mapping

AR Element	`ARPackage` *FrancaPortInterfaces*
AR Container	*FrancaModelPackage*
Attributes	shortName = "FrancaPortInterfaces"

Condition	—

Table 3.6: FrancaPortInterfaces mapping

AR Element	`ARPackage` *FrancaSwComponentTypes*
AR Container	*FrancaModelPackage*
Attributes	shortName = "FrancaSwComponentTypes"
Condition	—

Table 3.7: FrancaSwComponentTypes mapping

Franca version information is mapped to the AUTOSAR document revision.

[TR_FRANCA_00011] Franca version is mapped to AUTOSAR document revision
⌈An `FVersion` *fVersion* of a Franca element *fElement* is mapped to the document revision

```
docRevision = fVersion.major + "." + fVersion.minor
```

of the AUTOSAR element to which *fElement* is mapped.⌋*()*

3.3 Franca Types

AUTOSAR distinguishes application data types and implementation data types. Application data types allow to define all the data attributes which are needed from the application point of view, in order to exchange data between software components or between a software component and a measurement and calibration tool. Implementation data types correspond to the actual binary numbers handled by the programming language on the CPU. They contain concepts like pointers and unions which relate to the organization of data in memory and are not relevant for the application level. Implementation data types are in particular the source for the generation of C code. According to the goal of the Franca Integration stated in chapter 1 – to obtain an application level view of the integrated system – application data types are the appropriate target for the Franca-to-AUTOSAR translation. However, Franca IDL includes union types and type definitions as data type constructors; and these are not covered by the AUTOSAR application data type constructors.

AUTOSAR implementation data type constructors on the other hand comprise union types and type definitions, as well as the other Franca type constructors.[1] They could thus be used as targets for the Franca-to-AUTOSAR translation of data types. Since they are used for code generation, however, their specification requires many more details than given in a Franca model. Moreover, if a Franca model shall be used also for calibration or measurement purposes the expressiveness of AUTOSAR application data types is needed again.

[1] The only exception are map types, that are beyond the expressiveness of AUTOSAR. They are not translated.

For these reasons the Franca-to-AUTOSAR translation attempts to map a Franca data type to

- an AUTOSAR application data type,
- an AUTOSAR implementation data type,
- and an AUTOSAR data type mapping that relates these two data types.

As mentioned above it may happen that no AUTOSAR application data type is generated for a Franca data type. An AUTOSAR implementation data type is always generated, except for Franca map types that are not translated at all. When both an application data type and an implementation data type are generated they are related by a data type map.

[TR_FRANCA_00015] Application and implementation data type of a Franca data type are related ⌈When a Franca data type is mapped to an AUTOSAR `ApplicationDataType` *appType* and an AUTOSAR `ImplementationDataType` *implType* then *appType* and *implType* are related by a `DataTypeMap`. The data type maps that arise from the translation of a Franca model are collected in one global `DataTypeMappingSet` that is contained in the package *FrancaDataTypeMaps* defined in [TR_FRANCA_00010].⌋*()*

The translation of Franca data types to AUTOSAR data types induces a mapping *ARType* that is defined as follows.

[TR_FRANCA_00016] Mapping from Franca data types to AUTOSAR data types ⌈Let *fDataType* be a Franca data type.

1. If *fDataType* can be translated to an AUTOSAR application data type *arAppDataType* then

   ARType(*fDataType*) = *arAppDataType*.

2. If *fDataType* cannot be translated to an AUTOSAR application data type but to an AUTOSAR implementation data type *arImplDataType* then

   ARType(*fDataType*) = *arImplDataType*.

3. If *fDataType* can neither be translated to an AUTOSAR application data type nor to an AUTOSAR implementation data type then

   ARType(*fDataType*) is undefined.

⌋*()*

3.3.1 Franca Type Collections

A Franca type collection is a container for Franca data types, corresponding to an AUTOSAR package. According to the discussion above, each type collection yields

two packages, one for the generated application data types and one for the generated implementation data types.

[TR_FRANCA_00090] Franca type collection is mapped to AUTOSAR packages ⌈An `FTypeCollection` *fTypeCollection* is mapped to a sub-package of the application type package *FrancaApplicationDataTypes* and a sub-package of the implementation type package *FrancaImplementationDataTypes* defined in [TR_FRANCA_00010] as described in Table 3.8 and Table 3.9⌋*()*

AR Element	`ARPackage` *applicationTypeCollectionPackage*
AR Container	*FrancaApplicationDataTypes*
Attributes	shortName = *fTypeCollection*.name
Condition	—

Table 3.8: fTypeCollection mapping to the application type package

AR Element	`ARPackage` *implementationTypeCollectionPackage*
AR Container	*FrancaImplementationDataTypes*
Attributes	shortName = *fTypeCollection*.name
Condition	—

Table 3.9: fTypeCollection mapping to the implementation type package

3.3.2 Primitive Types

Franca IDL has a set of predefined, so called primitive, data types: integers, floats, Boolean values, and strings. Whereas integers, floats, and booleans are covered by the AUTOSAR platform types, strings have to be encoded. In the following we define the application and implementation data types that are generated by the Franca-to-AUTOSAR translation to represent the primitive types of Franca IDL. Both application and implementation types obtain the short names given in table 3.10.

[TR_FRANCA_00426] Franca primitive types and the corresponding AUTOSAR platform types ⌈Table 3.10 lists the supported Franca primitive types and the corresponding AUTOSAR platform types.⌋ *()*

Franca primitive type	AUTOSAR short name
UInt8	uint8
Int8	sint8
UInt16	uint16
Int16	sint16
UInt32	uint32
Int32	sint32
UInt64	uint64
Int64	sint64
Boolean	boolean
Float	float32
Double	float64

Table 3.10: Short names of AUTOSAR data types corresponding to Franca primitive types

[TR_FRANCA_00100] Primitive Type is mapped to `ApplicationPrimitiveDataType` ⌈A Franca primitive type is mapped to an `ApplicationPrimitiveDataType` with the categories and data properties as defined in Table 3.11.⌋ *()*

Franca Type	AR Category	AR Property
UInt8	VALUE	data constraint: lower limit = 0, upper limit = 255
Int8	VALUE	data constraint: lower limit = -128, upper limit = 127
UInt16	VALUE	data constraint: lower limit = 0, upper limit = 65535
Int16	VALUE	data constraint: lower limit = -32768, upper limit = 32767
UInt32	VALUE	data constraint: lower limit = 0, upper limit = 2^{32}
Int32	VALUE	data constraint: lower limit = -(2^{31}), upper limit = 2^{31}-1
UInt64	VALUE	data constraint: lower limit = 0, upper limit = 2^{64}
Int64	VALUE	data constraint: lower limit = -(2^{63}), upper limit = 2^{63}-1
Boolean	BOOLEAN	data constraint: lower limit = 0, upper limit = 1
Float	VALUE	—
Double	VALUE	—
String	STRING	maximum text size default = 256, can be redefined in the Franca deployment definition

Table 3.11: Categories and properties of AUTOSAR application data types corresponding to Franca primitive types

Application primitive data types are defined directly by the metaclass ApplicationPrimitiveDataType in AUTOSAR. To express that an implementation data type represents a primitive type in AUTOSAR its name must coincide with one of the AUTOSAR platform types (see [7]) and it must be associated to a base type that does not have a native declaration. This is reflected in the following definition of the translation of Franca primitive data types to AUTOSAR implementation types.

[TR_FRANCA_00110] Primitive Type is mapped to ImplementationDataType corresponding to AUTOSAR platform type ⌈A Franca primitive type, except the primitive type String, is mapped to an ImplementationDataType with the short name defined in table described by [TR_FRANCA_00426]. The base type of each of these implementation data types must not have a native declaration.⌋*()*

The primitive type String of Franca is interpreted as an array of characters. Franca does not defined whether strings have a fixed or a variable size. This can be defined in a Franca deployment definition. Strings of fixed size can represented directly as fixed size arrays in AUTOSAR. Arrays of variable size are encoded in AUTOSAR as structures with two elements: an integer field that defines the actual size of an array and an array field that contains the array itself. The latter also contains the maximal size of the array instances.

[TR_FRANCA_00120] Primitive type String of fixed size is mapped to AUTOSAR array implementation data type ⌈If the property *FixedStringLength* of the Franca primitive type String is set to *true* and the property *MaxStringLength* is *n* then the type String is mapped to the ImplementationDataType *stringImplType* defined as follows.

- *stringImplType*.shortName = *String*
- *stringImplType*.category = ARRAY
- *stringImplType*.subElement = *subElement*, with

- *subElement*.shortName = *Char*
- *subElement*.category = VALUE
- *subElement*.arraySize = n
- *subElement*.arraySizeSemantics = fixedSize

The default values of the two String properties are *FixedStringLength = false* and *MaxStringLength = 256.*⌋*()*

[TR_FRANCA_00121] Primitive type String of variable size is mapped to AUTOSAR structure implementation data type ⌈If the property *FixedStringLength* of the Franca primitive type String is not set (which means that is has the default value *false*) and the property *MaxStringLength* is *n* then the type String is mapped to the `ImplementationDataType` *stringImplType* defined as follows.

- *stringImplType*.shortName = *String*
- *stringImplType*.category = STRUCTURE
- *stringImplType*.subElements = {*size*, *chars*}

with the `ImplementationDataTypeElement`s *size* and *chars*

- *size*.shortName = *size*
- *size*.category = TYPE_REFERENCE
- *size*.swDataDefProps.implementationDataType = *uint8*
- *chars*.shortName = *chars*
- *chars*.category = ARRAY
- *chars*.subElement = *char*, with
- *char*.shortName = *char*
- *char*.category = TYPE_REFERENCE
- *char*.arraySize = n
- *char*.arraySizeSemantics = variableSize
- *char*.swDataDefProps.implementationDataType = *uint8*

The default value *MaxStringLength* is *256.*⌋*()*

3.3.3 Franca Inline Arrays

The types of method and broadcast arguments, attributes, and fields of union and structure types can be defined in Franca as inline arrays. That means that instead of an explicitly defined array type the inline notation

```
t[] element
```

can be used.

Since AUTOSAR does not support inline arrays the implicitly defined Franca array types have to be translated to explicit AUTOSAR application and implementation array types. This is achieved as specified in [TR_FRANCA_00200], [TR_FRANCA_00205], and [TR_FRANCA_00206].

In order to recover the original Franca model when the AUTOSAR description is translated back to Franca these array types are annotated with special data in the Franca special data group. Since this does not affect the semantics of the translation but only the syntactical representation it is not further specified here.

In the following we do not explicitly indicate the treatment of inline arrays but take it for granted that inline arrays are translated to explicitly defined AUTOSAR array types.

3.3.4 User-defined Types

Franca's user defined types comprise compound types like arrays, structures, and unions. The translation to AUTOSAR data types is defined in such a way that each type is either translated completely or not at all. Consider for instance a Franca array type whose elements are typed by a union type. A Franca array type can be translated to an AUTOSAR application data type; a Franca union type, however, cannot be translated to an AUTOSAR application data type. Therefore the above mentioned example of a Franca array of unions is not translated to an AUTOSAR application data type.

On the other hand, both array and union types can be translated to AUTOSAR implementation data types. Therefore also the Franca array of union type can be translated to an AUTOSAR implementation data type.

The only Franca data type that cannot be translated to an AUTOSAR data type at all is the map type. If this occurs in a compound type the whole compound type is also not mapped to any AUTOSAR data type.

3.3.4.1 Mapping to Application Data Types

[TR_FRANCA_00200] Application Array Type ⌈An `FArrayType` *fArrayType* is mapped to the AUTOSAR `ApplicationArrayDataType` *arArrayType* defined by

- *arArrayType*.shortName = *fArrayType*.name
- *arArrayType*.category = ARRAY
- *arArrayType*.element = *element* defined by
- *element*.shortName = *fArrayType*.name + "element"

- *element*.maxNumberOfElements = *fArrayType*.ArraySize
- *element*.arraySizeSemantics = fixedSize if *fArrayType*.ArrayFixedSize == true
  element.arraySizeSemantics = variableSize if *fArrayType*.ArrayFixedSize == false
- *element*.type = *ARType*(*fArrayType*.elementType)
- *element*.category = *ARType*(*fArrayType*.elementType).category

The values *fArrayType*.ArraySize and *fArrayType*.ArrayFixedSize are defined in the deployment definition of the Franca model that contains the data type.

If the *ARType*(*fArrayType*.elementType) is undefined then also the translation of *fArrayType* is not defined.⌋*()*

[TR_FRANCA_00210] Application Enumeration Type ⌈An `FEnumerationType` *fEnumerationType* is translated to the `ApplicationPrimitiveDataType` *arEnumerationType* defined by

- *arEnumerationType*.shortName = *fEnumerationType*.name
- *arEnumerationType*.category = VALUE

The set of `FEnumerator`s of *fEnumerationType* is mapped to a `CompuMethod` as defined in [4] [TPS_SWCT_01562].⌋*()*

[TR_FRANCA_00220] Application Structure Type ⌈An `FStructType` *fStructType* is mapped to the AUTOSAR `ApplicationRecordDataType` *arStructType* defined by

- *arStructType*.shortName = *fStructType*.name
- *arStructType*.category = STRUCTURE

and for each `FField` *fField* an `ApplicationRecordElement` *recordElement* defined by

- *recordElement*.shortName = *fField*.name
- *recordElement*.type = *ARType*(*fField*.type)

⌋*()*

[TR_FRANCA_00230] Application Union Type ⌈An `FUnionType` *fUnionType* is not mapped to an AUTOSAR application data type.⌋*()*

[TR_FRANCA_00240] Application Type Definition ⌈An `FTypeDef` *fTypeDef* is not mapped to an AUTOSAR application data type.⌋*()*

[TR_FRANCA_00250] Application Map Type ⌈An `FMapType` *fMapType* is not mapped to an AUTOSAR application data type.⌋*()*

3.3.4.2 Mapping to Implementation Data Types

Analogous to the distinction of fixed size strings ([TR_FRANCA_00120]) and variable size strings ([TR_FRANCA_00121]) the translation of Franca array types to AUTOSAR implementation data types distinguishes array types of fixed and variable size.

[TR_FRANCA_00205] Implementation Array Type of fixed size ⌈An `FArrayType` *fArrayType* whose property *ArrayFixedSize* is set to *true* is mapped to the `ImplementationDataType` *arArrayType* defined by

- *arArrayType*.shortName = *fArrayType*.name
- *arArrayType*.category = ARRAY
- *arArrayType*.subElement = *subElement* defined by
- *subElement*.shortName = *fArrayType*.name + "_elements"
- *subElement*.category = TYPE_REFERENCE
- *subElement*.arraySize = *fArrayType*.ArraySize
- *subElement*.arraySizeSemantics = fixedSize
- *subElement*.swDataDefProps.implementationDataType = *ARType*(*fArrayType*.elementType)

where *fArrayType*.ArraySize and *fArrayType*.ArrayFixedSize are defined in the deployment definition of the Franca model. The default value of *fArrayType*.ArrayFixedSize is *false*.

If *ARType*(*fArrayType*.elementType) is undefined *fArrayType* is not translated.⌋*()*

An arrays of variable size is represented in AUTOSAR at the implementation type level as a structure whose first element is an integer that denotes the actual size of the array and whose second element is the array itself.

[TR_FRANCA_00206] Implementation Array Type of variable size ⌈An `FArrayType` *fArrayType* whose property *ArrayFixedSize* is not set (which means that it has the default value *false*) or is set to *false* is mapped to the `ImplementationDataType` *stringImplType* defined as follows.

- *stringImplType*.shortName = *fArrayType*.name
- *stringImplType*.category = STRUCTURE
- *stringImplType*.subElements = {*size*, *array*}

with the `ImplementationDataTypeElement`s *size* and *array*

- *size*.shortName = *size*
- *size*.category = TYPE_REFERENCE
- *size*.swDataDefProps.implementationDataType = *uint8*

- *array*.shortName = *array*
- *array*.category = ARRAY
- *array*.subElement = *array_element*, with
- *array_element*.shortName = *array_element*
- *array_element*.category = TYPE_REFERENCE
- *array_element*.arraySize = *fArrayType*.ArraySize
- *array_element*.arraySizeSemantics = variableSize
- *array_element*.swDataDefProps.implementationDataType = *ARType*(*fArrayType*.type)

⌋()

[TR_FRANCA_00215] Implementation Enumeration Type ⌈An `FEnumerationType` *fEnumerationType* is translated to the `ImplementationDataType` *arEnumerationType* defined by

- *arEnumerationType*.shortName = *fEnumerationType*.name
- *arEnumerationType*.category = VALUE

The set of `FEnumerator`s of *fEnumerationType* is mapped to an `CompuMethod` as defined in [4] [TPS_SWCT_01562].⌋()

[TR_FRANCA_00225] Implementation Structure Type ⌈An `FStructType` *fStructType* is mapped to the `ImplementationDataType` *arStructType* defined by

- *arStructType*.shortName = *fStructType*.name
- *arStructType*.category = STRUCTURE

and for each `FField` *fField* of *fStructType* an `ImplementationDataTypeElement` *subElement* defined by

- *subElement*.shortName = *fField*.name
- *subElement*.category = TYPE_REFERENCE
- *subElement*.swDataDefProps.implementationDataType = *ARType*(*fField*.type)

If *ARType*(*fField*.type) is undefined *fStructType* is not translated.⌋()

[TR_FRANCA_00235] Implementation Union Type ⌈An `FUnionType` *fUnionType* is mapped to the `ImplementationDataType` *arUnionType* defined by

- *arUnionType*.shortName = *fUnionType*.name
- *arUnionType*.category = UNION

and for each `FField` *fField* of *fUnionType* an `ImplementationDataTypeElement` *subElement* defined by

- *subElement*.shortName = *fField*.name
- *subElement*.category = TYPE_REFERENCE
- *subElement*.swDataDefProps.implementationDataType = *ARType*(*fField*.type)

If *ARType*(*fField*.type) is undefined *fUnionType* is not translated.⌋*()*

[TR_FRANCA_00245] Implementation Type Definition ⌈An `FTypeDef` *fTypeDef* is mapped to the `ImplementationDataType` *arTypeDef* defined by

- *arTypeDef*.shortName = *fTypeDef*.name
- *arTypeDef*.category = TYPE_REFERENCE
- *arTypeDef*.swDataDefProps.implementationDataType = *ARType*(*fTypeDef*.actualType)

If *ARType*(*fTypeDef*.actualType) is not defined *fTypeDef* is not translated.⌋*()*

[TR_FRANCA_00255] Implementation Map Type ⌈An `FMapType` *fMapType* is not mapped to an AUTOSAR implementation data type.⌋*()*

3.3.5 Type Inheritance

Franca IDL allows type inheritance for enumerations, structures, and unions. Since AUTOSAR does not support inheritance the Franca type definitions have to be resolved when they are translated to AUTOSAR. That means that the resulting AUTOSAR type of a Franca enumeration directly contains all literals that are directly or indirectly contained in the Franca enumeration via its chain of base types. Analogously, the translation of a Franca structure or union type contains all fields that are directly or indirectly defined for the type.

This resolution does not change the semantics of the data types; however, it affects their syntactical representation. In order to be able to reconstruct the original Franca data type definition as close as possible when inverting the translation to AUTOSAR, the target AUTOSAR data types and their elements are annotated. The AUTOSAR means for that purpose are special data. A specific special data group with the gid *Franca_Transformation* is introduced that contains this annotation. None of the data contained in this special data group affects the semantics of the AUTOSAR software component description that results from the translation. Only information on the syntactic structuring is represented by this special data.

3.4 Franca Interfaces

3.4.1 Franca Interfaces

A single Franca interface may contain methods, attributes, and broadcasts. The corresponding elements on the AUTOSAR side are operations and data elements. Since an AUTOSAR operation can only be contained in a client server interface and an AUTOSAR data element can only be contained in a sender receiver interface, at least two AUTOSAR interfaces must be generated for one Franca interface. Franca IDL supports fire-and-forget methods that are mapped to data elements (sender receiver communication) instead of operations (client server communication). A fire-and-forget method offered by a Franca instance is called by an AUTOSAR component prototype in that the latter sends the corresponding data element to the Franca instance. As opposed to the methods and broadcasts, that are provided by a Franca instance that implements the correpsonding interface, the fire-and-forget methods – interpreted as data elements – are required by the Franca instance. This is reflected in the definition of the corresponding ports ([TR_FRANCA_00310]) and leads to the definition of a third AUTOSAR port interface corresponding to a Franca interface to represent the fire-and-forget methods.

The rules given below essentially define that

- a Franca method is mapped to an AUTOSAR operation, with the exception of a Franca fire-and-forget method that is mapped to an AUTOSAR data element
- a Franca attribute is mapped to a getter operation, a setter operation, and a notification data element,
- a Franca broadcast is mapped to an AUTOSAR data element.

The getter operation corresponding to a Franca attribute always exists. If the Franca flags *readonly* or *noSubscriptions* are set, the generation of the setter operation and the notification data elements respectively are prohibited. Thus if all attributes are read-only and not-subscribable and there are no broadcasts, no data elements will be generated. In this case also no provided sender receiver interface is generated. If there are no fire-and-forget methods no required sender receiver interface is generated.

[TR_FRANCA_00020] Franca interface is mapped to AUTOSAR client server interface and AUTOSAR sender receiver interfaces ⌈An `FInterface` *fInterface* is mapped to AUTOSAR interfaces as described in Table 3.12, Table 3.13 and Table 3.14.⌋*()*

AR Element	`ClientServerInterface` *FrancaProvOperationsInterface*
AR Container	*FrancaPortInterfaces*
Attributes	shortName = "prov_operations_" + *fInterface*.name
Condition	—

Table 3.12: Franca Interface mapping to `ClientServerInterface`

AR Element	`SenderReceiverInterface` *FrancaProvDataElementsInterface*

AR Container	*FrancaPortInterfaces*
Attributes	shortName = "prov_dataElements_" + *fInterface*.name
Condition	*fInterface* has at least one subscribable attribute that is not read-only or at least one broadcast.

Table 3.13: FrancaProvDataElementsInterface mapping to `SenderReceiverInterface`

AR Element	`SenderReceiverInterface` *FrancaReqDataElementsInterface*
AR Container	*FrancaPortInterfaces*
Attributes	shortName = "req_dataElements_" + *fInterface*.name
Condition	*fInterface* has at least one fire-and-forget method.

Table 3.14: FrancaReqDataElementsInterface mapping to `SenderReceiverInterface`

3.4.2 Franca Methods

A Franca method is mapped to an AUTOSAR client server operation. An exception are fire-and-forget methods that are consumed when called but do not deliver a return value. They are mapped to AUTOSAR data elements.

[TR_FRANCA_00030] Franca method is mapped to AUTOSAR client server operation ⌈An `FMethod` *fMethod* that is not a fire-and-forget method is mapped to the `ClientServerOperation` as described in Table 3.15.⌋*()*

AR Element	`ClientServerOperation` *csOperation*
AR Container	*FrancaProvOperationsInterface*
Attributes	shortName = *fMethod*.name
Condition	*fMethod* is not a fire-and-forget method.

Table 3.15: Franca method mapping

[TR_FRANCA_00031] Franca fire-and-forget method is mapped to AUTOSAR variable data prototype ⌈An `FMethod` *fMethod* whose *fireAndForget*-flag is set to *true* is mapped to the `VariableDataPrototype` as described in Table 3.16.⌋*()*

AR Element	`VariableDataPrototype` *srDataElement*
AR Container	*FrancaReqDataElementsInterface*
Attributes	shortName = *fMethod*.name
Condition	*fMethod* is a fire-and-forget method.

Table 3.16: Franca fire and forget method mapping

The type of the data element *srDataElement* is the structure type *fMethod_type* whose elements correspond to the types of the input arguments of *fMethod*. Depending on the types of the input arguments *fMethod_type* is either an `ApplicationRecordDataType` or an `ImplementationDataType`.

If *ARType*(*inArg*) is defined for each `FArgument` *inArg* contained in *fMethod*.inArgs and yields an `ApplicationDataType` then *fMethod_type* is the `ApplicationRecordDataType` defined as follows.

- *fMethod_type*.shortName = *fMethod*.name + "_type"
- *fMethod_type*.category = STRUCTURE

For each *inArg* contained in *fMethod*.inArgs the type *fMethod_type* contains an `ApplicationRecordElement` *recordElement* defined by

- *recordElement*.shortName = *inArg*.name
- *recordElement*.type = *ARType*(*inArg*)

If *ARType*(*inArg*) is defined for each `FArgument` *inArg* of *fMethod*.inArgs and at least one of them yields an `ImplementationDataType` then *fMethod_type* is the `ImplementationDataType` defined as follows.

- *fMethod_type*.shortName = *fMethod*.name + "_type"
- *fMethod_type*.category = STRUCTURE

For each *inArg* contained in *fMethod*.inArgs the type *fMethod_type* contains an `ImplementationDataTypeElement` *recordElement* defined by

- *recordElement*.shortName = *inArg*.name
- *recordElement*.category = TYPE_REFERENCE
- *recordElement*.swDataDefProps.implementationDataType = *ARType*(*inArg*)

If *ARType*(*inArg*) is undefined for at least one `FArgument` *inArg* of *fMethod*.inArgs then the the fire-and-forget method *fMethod* is not mapped.

A Franca argument of a method is mapped to an AUTOSAR argument data prototype. The translation of Franca arguments of broadcasts is defined below.

[TR_FRANCA_00040] Franca argument of a method is mapped to AUTOSAR argument data prototype ⌈An `FArgument` *fArgument* is mapped to an `ArgumentDataPrototype` if it is an argument of a method as described in Table 3.17.⌋ *()*

AR Element	`ArgumentDataPrototype` *arg*
AR Container	*csOperation*
Attributes	shortName = *fArgument*.name direction = ArgumentDirectionEnum.IN if *arg* is an input argument direction = ArgumentDirectionEnum.OUT if *arg* is an output argument
Condition	The method that contains *fArgument* as an input or output argument is mapped to the AUTOSAR client server operation *csOperation*.

Table 3.17: Franca argument of a method mapping

3.4.3 Franca Attributes

A Franca attribute is mapped to a getter operation, a setter operation, and a notification data element. The generation of the setter operation and the notification data element depends on the flags that are set for the attribute.

[TR_FRANCA_00050] Franca attribute is mapped to AUTOSAR client server operations and data prototypes ⌈An `FAttribute` *fAttribute* is mapped to AUTOSAR client server operations and data elements according to Table 3.18, Table 3.19, Table 3.20.⌋*()*

AR Element	`ClientServerOperation` *getter*
AR Container	*FrancaProvOperationsInterface*
Attributes	shortName = "get_" + *fAttribute*.name
Condition	—

Table 3.18: Getter mapping

AR Element	`ClientServerOperation` *setter*
AR Container	*FrancaProvOperationsInterface*
Attributes	shortName = "set_" + *fAttribute*.name
Condition	*fAttribute* is not read-only.

Table 3.19: Setter mapping

AR Element	`VariableDataPrototype` *notification*
AR Container	*FrancaProvDataElementsInterface*
Attributes	shortName = "notify_" + *fAttribute*.name type = *ARType*(*fAttribute*.type)
Condition	*fAttribute* is subscribable.

Table 3.20: Notification mapping

3.4.4 Franca Broadcasts

A Franca broadcast is mapped to an AUTOSAR data element. The type of the data element is a structure type whose elements are determined by the out-arguments of the broadcast.

[TR_FRANCA_00070] Franca broadcast is mapped to AUTOSAR variable data prototype ⌈An `FBroadcast` *fBroadcast* is mapped to the variable data prototype according to Table 3.21.⌋*()*

AR Element	VariableDataPrototype *broadcast*
AR Container	*FrancaProvDataElementsInterface*
Attributes	shortName = "broadcast_" + *fBroadcast*.name type: AUTOSAR struct-type whose fields are given by the names and the Franca-to-AUTOSAR type translations of the types of the out-arguments of *fBroadcast*.
Condition	—

Table 3.21: Broadcast mapping

3.4.5 Interface Inheritance

Franca interface inheritance is handled in the same way as Franca data type inheritance. The translation generates target elements for all elements that are directly or indirectly contained in a Franca interface according to its inheritance hierarchy. The elements that are indirectly contained are annotated by special data in the *Franca_Transformation* special data group. Using this annotation they can be handled appropriately by the inverse translation from AUTOSAR to Franca IDL.

3.5 Franca Connector

A Franca Connector declares Franca instances and connections between Franca instances and AUTOSAR component prototypes. A Franca instance implements a set of Franca interfaces. This set may also be empty, which can be used to declare Franca instances that use AUTOSAR operations, but whose provided interfaces are not relevant for the Franca Integration.

A Franca instance is translated to an AUTOSAR component prototype. The type of this component prototype is determined by the list of interfaces that are implemented by the Franca instance. For each list of implemented interfaces that appears in the instance declaration part of the Franca Connector one AUTOSAR ApplicationSwComponentType is generated. It contains, for each Franca interface in the list, three ports. The first one is a provided port typed by a client server interface that contains operations representing the methods and the getter and setter operations for the attributes contained in the Franca interface. The second one is also a provided port, typed by a sender receiver interface that contains data elements representing the attribute change notifications and the broadcasts contained the Franca interface. The third one is a required port, also typed by a sender receiver interface, which contains data elements representing the fire-and-forget methods of the Franca interface.

[TR_FRANCA_00300] Franca instance is mapped to AUTOSAR component prototype and AUTOSAR application component type ⌈A Franca instance *g* that implements the Franca Interfaces *F1*, ..., *Fn* is mapped to a SwComponentPrototype *componentInstance* with *shortName g*.

Depending on the links of the Franca Connector in which the *componentInstance g* appears the CompositionSwComponentType that contains *g* is determined. If there is

a link that contains the *g* then the container of *g* is the container of the `CompositionSwComponentType` that also contains the `SwComponentPrototype` at the other end of the link. If none of the links contains the *componentInstance* then its container is a newly created `CompositionSwComponentType`.⌋*()*

The type of *g* is given by the following `ApplicationSwComponentType` *componentType*.

AR Element	`ApplicationSwComponentType` *componentType*
AR Container	*FrancaSwComponentTypes*
Attributes	shortName = "type_" + *g*
Condition	The type for the list of Franca Interfaces implemented by *g* has not yet been generated.

Table 3.22: Franca instance mapping

Each Franca interface that is implemented by a Franca instance induces ports for the type *componentType* of the component instance defined above. There are two provided ports for the methods and broadcasts respectively of the Franca interface, and one required port for the fire-and-forget methods. Recall that the latter are mapped to data elements that are sent to the component instance.

[TR_FRANCA_00310] Franca interface implemented by a Franca instance yields AUTOSAR ports of *componentType* ⌈Each Franca interface *F* implemented by a Franca instance *g* generates `PortPrototype`s described in Table 3.23, Table 3.24, Table 3.25 for the *componentType* of *g* defined in [TR_FRANCA_00300].⌋*()*

AR Element	`PPortPrototype` *csProvPort*
AR Container	*componentType*
Attributes	shortName = "csProvPort_" + *fInterface*.name, providedInterface = *FrancaProvOperations*
Condition	—

Table 3.23: Franca Interface mapping to a ClientServer `PPortPrototype`

AR Element	`PPortPrototype` *srProvPort*
AR Container	*componentType*
Attributes	shortName = "srProvPort_" + *fInterface*.name, providedInterface = *FrancaProvDataElements*
Condition	*FrancaProvDataElements* exists.

Table 3.24: Franca Interface mapping to a SenderReceiver `PPortPrototype`

AR Element	`RPortPrototype` *srReqPort*
AR Container	*componentType*
Attributes	shortName = "srReqPort_" + *fInterface*.name, requiredInterface = *FrancaReqDataElements*
Condition	*srInterface* exists.

Table 3.25: Franca Interface mapping to a SenderReceiver `RPortPrototype`

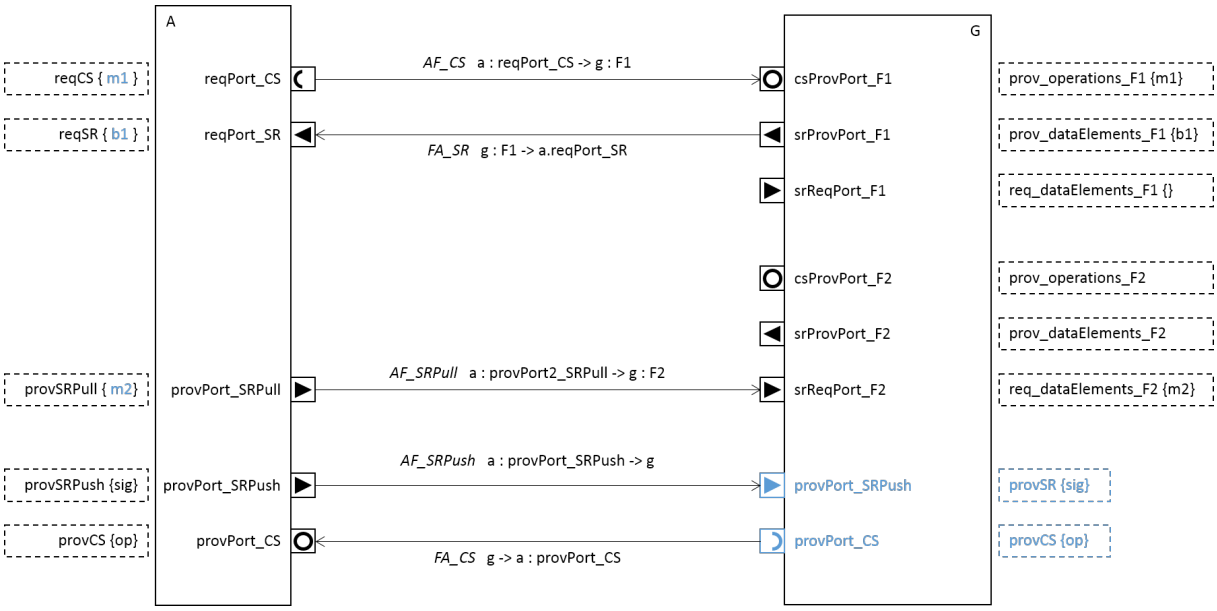

Figure 3.1: Translation of Franca links to AUTOSAR

Figure 3.1 indicates the translation of the links contained in a Franca Connector. Essentially, a Franca link generates an AUTOSAR assembly software connector. The direction of the link – AUTOSAR-to-Franca or Franca-to-AUTOSAR – and the type of the AUTOSAR port of the link – client-server-interface or sender-receiver-interface – determine the context components and the target ports of the assembly's provider and requester. The translation of the four kinds of links (see section 2.2) is discussed in the following.

Throughout the discussion we use the names for ports, interfaces, and links introduced in Figure 3.1. We first consider the case in which each Franca instance can be placed into the same container (`CompositionSwComponentType`) as the AUTOSAR component prototype it is linked to. This holds if all links that contain the Franca instance have AUTOSAR component prototypes on the other side that are contained in one and same container. This is then also the container of the `AssemblySwConnector`s that are generated for the links. The more general case is discussed in subsection 3.5.6.

3.5.1 AUTOSAR-to-Franca Client Server Link

An AUTOSAR-to-Franca client server link

autosar_port *a : reqPort_CS* → **franca_instance** *g : F1*

is translated to an `AssemblySwConnector` *assemblyConnector* with the `PPortInCompositionInstanceRef` *assemblyProvider* and `RPortInCompositionInstanceRef` *assemblyRequester* defined as follows.

provided context component the *componentInstance g* that is generated by the translation of the Franca component instance *g*.

provided target port the port *csProvPort_F1* that is generated by the translation of the Franca Interface *F1*

requested context component the `SwComponentPrototype` *a*.

requested target port the `RPortPrototype` *reqPort_CS*.

The `ClientServerInterface` *reqCS* that types *reqPort_CS* is updated as follows. For each `ClientServerOperation` *op* in the interface *prov_operations_F1* that types *csProvPort_F1* a copy of *op* is added to *reqCS*. That means that *reqCS* contains representations of all methods and getter/setter operations of *F1*.

An implementation of the transformation must ensure that the names of the client server operations in the updated interface *reqCS* are unique. If *reqCS* already contained an operation with the same name as an operation *op* carried over from *prov_operations_F1* then a new name – for instance the full qualified name of *op* – has to be generated for the copy. In addition to that a `ClientServerInterfaceMapping` that relates the two names has to be added and referenced by the *assemblyConnector*.

3.5.2 AUTOSAR-to-Franca Sender Receiver Link

An AUTOSAR-to-Franca sender receiver link

autosar_port *a : provPort_SRPush* → **franca_instance** *g*

is translated to an `AssemblySwConnector` *assemblyConnector* with the `PPortInCompositionInstanceRef` *assemblyProvider* and `RPortInCompositionInstanceRef` *assemblyRequester* defined as follows.

provided context component the `SwComponentPrototype` *a*.

provided target port the `PPortPrototype` *provPort_SRPush*.

requested context component the *componentInstance g* that is generated by the translation of the Franca component instance *g*.

requested target port a copy of *provPort_SRPush* that is attached to the `ApplicationSwComponentType` *componentType* generated by the translation of the Franca component instance *g*. If the *componentType* already contains a port with the same name as *provPort_SRPush* a new name has to be generated for the copy that is unique within the name space of *componentType*.

The type of the new RPortPrototype is the interface *provSRPush* that also types *provPort_SRPush*.

3.5.3 AUTOSAR-to-Franca Sender Receiver Link for Fire-And-Forget-Methods

An AUTOSAR-to-Franca sender receiver link

autosar_port *a : provPort_SRPull* → **franca_instance** *g : F2*

is translated to an AssemblySwConnector *assemblyConnector* with the PPortInCompositionInstanceRef *assemblyProvider* and RPortInCompositionInstanceRef *assemblyRequester* defined as follows.

provided context component the SwComponentPrototype *a*.

provided target port the PPortPrototype *provPort_SRPull*.

requested context component the *componentInstance g* that is generated by the translation of the Franca component instance *g*.

requested target port the RPortPrototype *srReqPort_F2*.

The SenderReceiverInterface *provSRPull* that types *provPort_SRPull* is updated as follows. For each VariableDataPrototype *m* in the interface *req_dataElements_F2* that types *srReqPort_F2* a copy of *m* is added to *provSRPull*. That means that *provSRPull* contains representations of all fire-and-forget methods of *F2*. If the interface *provSRPull* already contained a data element with the same name as *m* a new name has to be generated that is unique within the name space of *provSRPull*.

3.5.4 Franca-to-AUTOSAR Client Server Link

A Franca-to-AUTOSAR Client Server Link

franca_instance *g* → **autosar_port** *a : provPort_CS*

is translated to an AssemblySwConnector *assemblyConnector* with the PPortInCompositionInstanceRef *assemblyProvider* and RPortInCompositionInstanceRef *assemblyRequester* defined as follows.

provided context component the SwComponentPrototype *a*.

provided target port the PPortPrototype *provPort_CS*.

requested context component the *componentInstance* that is generated by the translation of the Franca component instance *g*.

requested target port a copy of *provPort_CS* is attached to the ApplicationSwComponentType *componentType* generated by the translation of the Franca component instance *g*. If *componentType* already contains a port with the same

name as *provPort_CS* a new name has to be generated that is unique within the name space of *componentType*.

The type of the new `RPortPrototype` is the interface *provCS* that also types *provPort_CS*.

3.5.5 Franca-to-AUTOSAR Sender Receiver Link

A Franca-to-AUTOSAR Attribute Link

franca_instance *g : F1* → **autosar_port** *a : reqPort_SR*

is translated to an `AssemblySwConnector` *assemblyConnector* with the `PPortInCompositionInstanceRef` *assemblyProvider* and `RPortInCompositionInstanceRef` *assemblyRequester* defined as follows.

provided context component the *componentInstance g* that is generated by the translation of the Franca component instance *g*.

provided target port the *srProvPort_F1* that is generated by the translation of the Franca Interface *F1*

requested context component the `SwComponentPrototype` *a*.

requested target port the `RPortPrototype` *reqPort_SR*.

The `SenderReceiverInterface` *reqSR* that types *reqPort_SR* is updated as follows. For each `VariableDataPrototype` *b* in the interface *prov_dataElements_F1* that types *srProvPort_F1* a copy of *b* is added to *reqSR*. That means that *reqSR* contains representations of all attribute change notifications and broadcasts of *F1*. If *reqSR* already contains a data element with the same name as *b* two cases have to be distinguished.

1. If *b* is already contained in *reqSR* due to the translation of another link with the same Franca side **franca_instance** *g : F1* as the currently considered link then no new copy is generated.
2. Otherwise the copy is added an a new name is generated that is unique within the name space of *reqSR*.

3.5.6 Connecting Instances in Disjoint Containers

The precondition for the definitions above has been that the Franca instance and the AUTOSAR component prototype of a link are contained in the same container (`CompositionSwComponentType`). As soon as there are two links connecting a Franca instance to two AUTOSAR component prototypes that are contained in different composition types this precondition no longer holds. In this case instead of a single assembly connector a chain of delegation and assembly connectors has to be generated.

Let *C0* be the least composition type that contains – via a chain of containment and type relations – the Franca instance *f* and the AUTOSAR component prototype *a*. Figure 3.2 shows a prototypical constellation. `CompositionSwComponentType` *C2* contains the AUTOSAR `SwComponentPrototype` *a*; `CompositionSwComponentType` *C1* contains a `SwComponentPrototype` *c2* whose type is *C2*, and *C0* contains a `SwComponentPrototype` *c1* whose type is *C1*. An analogous hierarchy whose for the containment of the Franca instance *f* is shown on the right side of the figure. In order to connect *a* and *f* the `DelegationSwConnector`s *del2_p* and *del1_p* have to added to the component types *C2* and *C1* respectively. In this step also new `PortPrototype`s have to be added to *C2* and *C1* as *proxies* of the start port *p* that shall be connected. In this step name clashes have to be avoided concerning both the new delegations and the new ports. Analogous delegations and ports have to be generated on the other side. Finally within *C0* the `AssemblySwConnector` *assembly* can be generated.

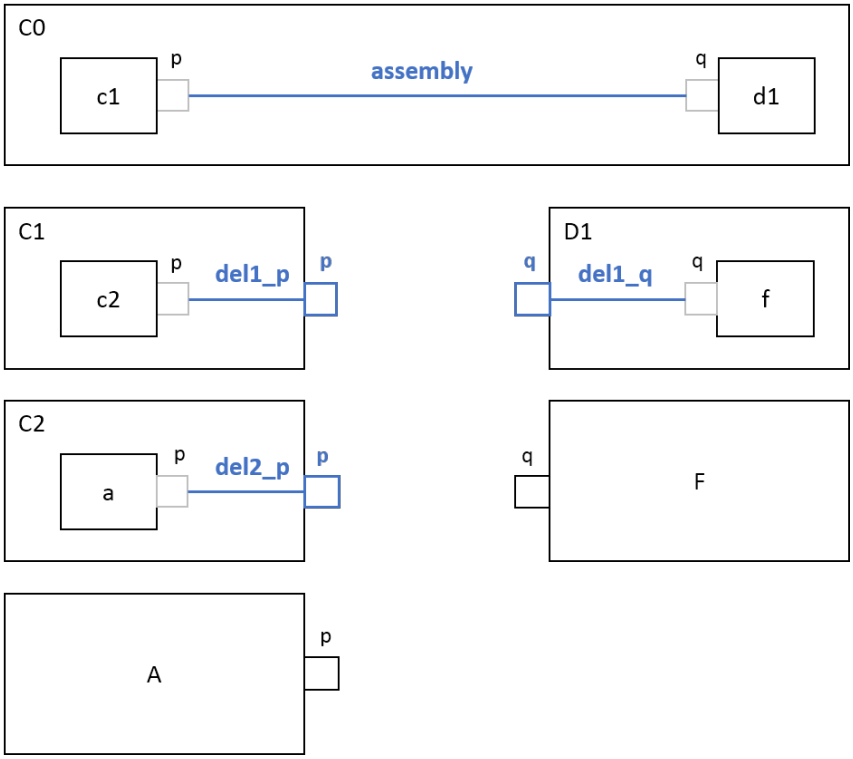

Figure 3.2: Connection of component prototypes in different composition component types.

4 AUTOSAR-to-Franca Translation

The AUTOSAR-to-Franca translation collects the data types and port interfaces from an AUTOSAR xml-file and brings them into the Franca IDL format. It is thus rather a filter than a translation.

4.1 Data Types

Franca does not distinguish application and implementation data types. Since all data type attributes that are relevant for a Franca model are covered by the AUTOSAR application data type attributes, in the first line application data types are considered. Whenever an AUTOSAR implementation data type is related to an application data type via a data type map the latter is considered as the representative of the implementation data type. That means that only the application data type is translated to Franca IDL. The implementation data type must be semantically compatible with the application data type and therefore yields no further information for the translation.

If an implementation data type is not related to any application data type, however, it will be translated. This will always be the case for union types and type definitions that are not available at the AUTOSAR application type level. Since it might happen that the AUTOSAR input does not contain data type maps the translation must be defined for all kinds of both implementation and application data types.

The translation selects the data types from the AUTOSAR input that have a representation in Franca IDL. AUTOSAR data types that do not fit into any of the patterns that are defined below are not translated.

[TR_FRANCA_00380] Mapping from AUTOSAR data types to Franca data types ⌈The translation of AUTOSAR data types to Franca data types induces a mapping *FType* that is defined as follows. Let *arDataType* be an AUTOSAR data type.

```
FType(arDataType) = fDataType
```

if *arDataType* is translated to the Franca data type *fDataType*;

```
FType(arDataType) is undefined
```

if *arDataType* cannot be translated to a Franca data type.⌋*()*

4.1.1 Platform Types

An AUTOSAR implementation data type is a platform type if its short name coincides with the short name of one of the AUTOSAR platform types defined in [7] and its base type has no native declaration. The name correspondence of Franca primitive types

and AUTOSAR platform types has been defined in table 3.10. Read from right to left this yields the mapping of AUTOSAR implementation types to Franca primitive types.

[TR_FRANCA_00390] Implementation platform type is mapped to primitive type ⌈If an AUTOSAR implementation type *implDataType* is a platform type it is mapped to the Franca primitive type defined by the name correspondence according to the table described in [TR_FRANCA_00426].⌋*()*

The corresponding mapping of application data types is also induced by the names. In this case the data properties defined in **TR_FRANCA_0100** are used to detect the application data types that correspond to Franca primitive types.

[TR_FRANCA_00395] Application data type with appropriate propterties is mapped to primitive type ⌈An AUTOSAR application type *applDataType* is mapped to the Franca primitive type *fPrimitiveType* if the category and properties of *applDataType* coincide with the ones stated in **TR_FRANCA_0100** and the short name of *applDataType* corresponds to the name of *fPrimitiveType* via the relation defined in table described by [TR_FRANCA_00426].⌋*()*

4.1.2 User-defined Types

4.1.2.1 Application Data Types

[TR_FRANCA_00400] Application array data type is mapped to Franca array type ⌈An `ApplicationArrayDataType` *appArrayType* is mapped to the `FArrayType` *fArrayType* defined by

- *fArrayType*.name = *appArrayType*.shortName
- *fArrayType*.elementType = *FType*(*arArrayType.element.type*)

If *FType*(*arArrayDataType.element.type*) is not defined then *appArrayType* is not translated.⌋*()*

[TR_FRANCA_00410] Application value data type is mapped to Franca enumeration type ⌈An `ApplicationPrimitiveDataType` *applicationPrimitiveType* of category VALUE is translated to the `FEnumerationType` *fEnumerationType* with *fEnumerationType*.name = *applicationPrimitiveType*.shortName if the algorithm for the detection of enumeration types defined in [3] can be applied. The latter also yields the `FEnumerator`s of *fEnumerationType*.⌋*()*

[TR_FRANCA_00420] Application record data type is mapped to Franca struct type ⌈An `ApplicationRecordDataType` *appRecordType* is mapped to the `FStructType` *fStructType* with *fStructType*.name = *appStructType*.shortName and for each `ApplicationRecordElement` *recordElement* in *arRecordType*.elements an `FField` *fField* with

- *fField*.name = *recordElement*.shortName
- *fField*.type = *FType*(*recordElement.type*)

If *FType*(*recordElement.type*) is not defined then *appRecordType* is not translated.⌋()

4.1.2.2 Implementation Data Types

[TR_FRANCA_00405] Array ⌈An `ImplementationDataType` *implDataType* of category ARRAY is mapped to the `FArrayType` *fArrayType* defined by

- *fArrayType*.name = *implArrayType*.shortName
- *fArrayType*.elementType = *FType*(*swDataDefProps*.implementationDataType)

where *swDataDefProps* is the `SwDataDefProps` of the (unique) sub-element of *implDataType*.

If *FType*(*swDataDefProp*.implementationDataType) is not defined then *implArrayType* is not translated.⌋()

[TR_FRANCA_00415] Value ⌈An `ImplementationDataType` *implDataType* of category VALUE is mapped to the `FEnumerationType` *fEnumerationType* with *fEnumerationType*.name = *implDataType*.shortName if the algorithm for the detection of enumeration types defined in [3] can be applied. The latter also yields the `FEnumerator`s of *fEnumerationType*.⌋()

[TR_FRANCA_00424] Struct representing an array ⌈Let *implDataType* be an `ImplementationDataType` of category STRUCTURE that matches the pattern defined in [TR_FRANCA_00206]. That means that *implDataType* contains exactly two *subElements*

- *size* which is an `ImplementationDataTypeElement` of category TYPE_REFERENCE that references the `ImplementationDataType` *uint8*, and
- *array* which is an `ImplementationDataTypeElement` of category ARRAY.

implDataType is mapped to the `FArrayType` that is obtained by the application of rule [TR_FRANCA_00405] to the `ImplementationDataTypeElement` *array*.⌋()

[TR_FRANCA_00425] Struct ⌈An `ImplementationDataType` *implDataType* of category STRUCTURE that does not match the pattern defined in [TR_FRANCA_00424] is mapped to the `FStructType` *fStructType* with *fStructType*.name = *implDataType*.shortName and for each `ImplementationDataTypeElement` *element* in *implDataType*.subElements an `FField` *fField* with

- *fField*.name = *element*.shortName
- *fField*.type = *FType*(*element*.swDataDefProps.implementationDataType)

If *FType*(*element*.swDataDefProps.implementationDataType) is not defined then *implDataType* is not translated.⌋()

[TR_FRANCA_00435] Union ⌈An `ImplementationDataType` *implDataType* of category UNION is mapped to the `FStructType` *fStructType* with *fStructType*.name = *implDataType*.shortName and for each `ImplementationDataTypeElement` *element* in *implDataType*.subElements an `FField` *fField* with

- *fField*.name = *element*.shortName
- *fField*.type = *FType*(*element*.swDataDefProps.implementationDataType)

If *FType*(*element*.swDataDefProps.implementationDataType) is not defined then *implDataType* is not translated.

⌋*()*

[TR_FRANCA_00445] Type Definition ⌈An `ImplementationDataType` *implDataType* of category TYPE_REFERENCE is mapped to the `FTypeDef` *fTypeDef* defined by

- *fTypeDef*.name = *implDataType*.shortName
- *fTypeDef*.actualType = *FType*(*implDataType*.swDataDefProps.implementationDataType)

If *FType*(*implDataType*.swDataDefProps.implementationDataType) is not defined then *implDataType* is not translated.⌋*()*

4.2 Port Interfaces

AUTOSAR port interfaces are mapped to Franca interfaces. In general an AUTOSAR operation is mapped to a Franca method and an AUTOSAR data element is mapped to a Franca broadcast. Only if the AUTOSAR input contains Franca special data Franca attributes will be recovered from the getter methods and fire-and-forget methods will be recovered from data elements. In the first case the corresponding setter methods and notifications will be ignored since they all represent the same Franca attribute that has already been derived from the getter method. In the second case a method is generated instead of a broadcast.

Franca special data is also used to identify AUTOSAR sender receiver interfaces that represents the same Franca interfaces as a client server interface. As defined in chapter 3 a Franca interface is translated to three interfaces: a client server interface for the methods, a sender receiver interface for the attribute change notifications and the broadcasts, and a sender receiver interface for the fire-and-forget methods. These AUTOSAR interfaces are accordingly mapped back to one Franca interface.

[TR_FRANCA_00500] Port Interface ⌈A `PortInterface` *arPortInterface* is mapped to an `FInterface` *fInterface* with the contents defined in the following rules.⌋*()*

[TR_FRANCA_00510] Client Server Operation ⌈A `ClientServerOperation` *csOperation* that is not tagged as Franca getter or setter method is mapped to the `FMethod` *fMethod* defined by

- *fMethod*.name = *csOperation*.shortName
- *fMethod*.inArgs is given by the translation of the *csOperation*.arguments that have the direction IN
- *fMethod*.outArgs is given by the translation of the *csOperation*.arguments that have the direction OUT

An `ArgumentDataPrototype` *arArgument* of *csOperation*.arguments is mapped to an `FArgument` *fArgument* of *fMethod* defined by

- *fArgument*.name = *arArgument*.shortName
- *fArgument*.type = *FType*(*arArgument*.type)

If *FType*(*arArgument*.type) is not defined *csOperation* is not translated.⌋()

[TR_FRANCA_00520] Variable Data Prototype ⌈A `VariableDataPrototype` *dataElement* that is not tagged as a Franca fire-and-forget methods is mapped to the `FBroadcast` *fBroadcast* defined by

- *fBroadcast*.name = "broadcast_" + *dataElement*.shortName
- *fBroadcast*.outArgs = (*fArgument*), the singleton list defined by
- *fArgument*.name = *dataElement*.shortName
- *fArgument*.type = *FType*(*dataElement*.type)

If *FType*(*dataElement*.type) is not defined *dataElement* is not translated.

If *dataElement* that tagged as a Franca fire-and-forget method it is mapped to the `FMethod` *fMethod* with *fMethod*.name = *dataElement*.shortName and the following `FArgument`s *fArgument* as inputs. For each `ApplicationRecordElement` or `ImplementationDataTypeElement` *recordElement* contained in the type of *dataElement* as defined in [TR_FRANCA_00031]

- *fArgument*.name = *recordElement*.shortName
- *fArgument*.type = *FType*(*recordElement*.type)

⌋()

4.3 Franca special data

As discussed in subsection 3.3.5 and subsection 3.4.5 inherited elements of Franca Models are annotated using Franca special data. During the Franca-to-AUTOSAR translation inherited elements are resolved and the completed models are translated to AUTOSAR. In order to receive a Franca Model that resembles the original as close as possible, in particular the inheritance structure has to be recovered. The two special data elements that are used for this purpose are the derived-tag that indicates that a Franca element is derived and the base-reference-tag that points to the base of a

Franca element. These two tags are evaluated in the AUTOSAR-to-Franca translation. When an AUTOSAR element has a derived-tag it is ignored. When an AUTOSAR element has a base-reference-tag the referenced element is added as *base*-attribute to the Franca element. This is done for all Franca elements that have a *base*-attribute.

A Examples

The following listings show the major parts of the Franca Integration example indicated in Figure 2.1. Listing A.1 shows the Franca model of the Franca interfaces *F1* and *F2*. The AUTOSAR XML-file that defines the component prototype *a* is shown in listing A.2. Listing A.3 shows the Franca Connector that defines the Franca instance*g* that implements the Franca interfaces *F1* and *F2* and its links to the AUTOSAR component prototype *a*.

Example A.1

```
interface F1 {
  method m1 {}
  broadcast b1 {
    out{
      UInt8 mb1
      UInt8 mb2
    }
  }
}

interface F2 {
  method m2 fireAndForget {}
}
```

Example A.2

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.
    org/2001/XMLSchema-instance" xsi:schemaLocation="http://autosar.org/
    schema/r4.0 autosar_4-1-1.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>autosar</SHORT-NAME>
      <ELEMENTS>
        <APPLICATION-PRIMITIVE-DATA-TYPE UUID="d2cf9cc2-9c01-3324-971d-738
            afdfabf20">
          <SHORT-NAME>UInt8</SHORT-NAME>
          <CATEGORY>VALUE</CATEGORY>
        </APPLICATION-PRIMITIVE-DATA-TYPE>
        <IMPLEMENTATION-DATA-TYPE UUID="45007175-9d62-360a-98a2-7054
            be84318d">
          <SHORT-NAME>UInt8Impl</SHORT-NAME>
          <CATEGORY>VALUE</CATEGORY>
          <SW-DATA-DEF-PROPS>
            <SW-DATA-DEF-PROPS-VARIANTS>
              <SW-DATA-DEF-PROPS-CONDITIONAL/>
            </SW-DATA-DEF-PROPS-VARIANTS>
          </SW-DATA-DEF-PROPS>
        </IMPLEMENTATION-DATA-TYPE>
        <DATA-TYPE-MAPPING-SET UUID="9293a76d-8ea6-3fff-9035-1afc9c97f086">
          <SHORT-NAME>dataTypeMappingSet</SHORT-NAME>
          <DATA-TYPE-MAPS>
```

```
      <DATA-TYPE-MAP>
        <APPLICATION-DATA-TYPE-REF DEST="APPLICATION-PRIMITIVE-DATA-
            TYPE">/autosar/UInt8</APPLICATION-DATA-TYPE-REF>
        <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE"
            >/autosar/UInt8Impl</IMPLEMENTATION-DATA-TYPE-REF>
      </DATA-TYPE-MAP>
    </DATA-TYPE-MAPS>
  </DATA-TYPE-MAPPING-SET>
  <CLIENT-SERVER-INTERFACE UUID="3734d142-7bb9-36d8-91b4-09bb0b575d42
      ">
    <SHORT-NAME>reqCS</SHORT-NAME>
    <IS-SERVICE>false</IS-SERVICE>
  </CLIENT-SERVER-INTERFACE>
  <SENDER-RECEIVER-INTERFACE UUID="04b5ab3e-601f-3386-9f7d-63
      a6f55e11c8">
    <SHORT-NAME>reqSR</SHORT-NAME>
    <IS-SERVICE>false</IS-SERVICE>
  </SENDER-RECEIVER-INTERFACE>
  <SENDER-RECEIVER-INTERFACE UUID="755e8152-0c87-3427-94dd-
      da407ab39759">
    <SHORT-NAME>provSRPull</SHORT-NAME>
    <IS-SERVICE>false</IS-SERVICE>
  </SENDER-RECEIVER-INTERFACE>
  <SENDER-RECEIVER-INTERFACE UUID="f9b76c74-bb67-30c1-bff7-2
      fe79bef5106">
    <SHORT-NAME>provSRPush</SHORT-NAME>
    <IS-SERVICE>false</IS-SERVICE>
    <DATA-ELEMENTS>
      <VARIABLE-DATA-PROTOTYPE UUID="693ee17a-86f2-34a6-bd36-
          d70148e3da35">
        <SHORT-NAME>sig</SHORT-NAME>
        <SW-DATA-DEF-PROPS>
          <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
              <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
          </SW-DATA-DEF-PROPS-VARIANTS>
        </SW-DATA-DEF-PROPS>
        <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">/autosar/
            UInt8</TYPE-TREF>
      </VARIABLE-DATA-PROTOTYPE>
    </DATA-ELEMENTS>
  </SENDER-RECEIVER-INTERFACE>
  <CLIENT-SERVER-INTERFACE UUID="56dc14c0-6550-30e5-9e62-a7d7e03e9150
      ">
    <SHORT-NAME>provCS</SHORT-NAME>
    <IS-SERVICE>false</IS-SERVICE>
    <OPERATIONS>
      <CLIENT-SERVER-OPERATION UUID="e877d6a3-1f83-3dec-b9b1-
          addef11e3ca8">
        <SHORT-NAME>op</SHORT-NAME>
      </CLIENT-SERVER-OPERATION>
    </OPERATIONS>
  </CLIENT-SERVER-INTERFACE>
  <APPLICATION-SW-COMPONENT-TYPE UUID="ded260d5-532f-3250-9227-
      ccdc1d0d0a2f">
```

```
          <SHORT-NAME>A</SHORT-NAME>
          <PORTS>
            <R-PORT-PROTOTYPE UUID="5cd37e72-4682-3fda-92fa-95a810de29c7">
              <SHORT-NAME>reqPort_CS</SHORT-NAME>
              <REQUIRED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">/
                 autosar/reqCS</REQUIRED-INTERFACE-TREF>
            </R-PORT-PROTOTYPE>
            <R-PORT-PROTOTYPE UUID="6330886b-e7e5-30ef-ada6-32324ce4d5a6">
              <SHORT-NAME>reqPort_SR</SHORT-NAME>
              <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/
                 autosar/reqSR</REQUIRED-INTERFACE-TREF>
            </R-PORT-PROTOTYPE>
            <P-PORT-PROTOTYPE UUID="b55ce023-9a76-37a1-a941-80f27a10f72e">
              <SHORT-NAME>provPort_SRPull</SHORT-NAME>
              <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/
                 autosar/provSRPull</PROVIDED-INTERFACE-TREF>
            </P-PORT-PROTOTYPE>
            <P-PORT-PROTOTYPE UUID="0673841e-a393-32d6-ad0b-d6dad901310c">
              <SHORT-NAME>provPort_SRPush</SHORT-NAME>
              <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/
                 autosar/provSRPush</PROVIDED-INTERFACE-TREF>
            </P-PORT-PROTOTYPE>
            <P-PORT-PROTOTYPE UUID="b009abfe-1538-3c83-b2cc-d596b65d8d11">
              <SHORT-NAME>provPort_CS</SHORT-NAME>
              <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">/
                 autosar/provCS</PROVIDED-INTERFACE-TREF>
            </P-PORT-PROTOTYPE>
          </PORTS>
        </APPLICATION-SW-COMPONENT-TYPE>
        <COMPOSITION-SW-COMPONENT-TYPE UUID="acb0061a-b365-32d4-b0e4-9
           fea3427044d">
          <SHORT-NAME>C</SHORT-NAME>
          <COMPONENTS>
            <SW-COMPONENT-PROTOTYPE UUID="2dcf4e8f-1ca0-3270-b247-9278
               f627d8ee">
              <SHORT-NAME>a</SHORT-NAME>
              <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/autosar/A</
                 TYPE-TREF>
            </SW-COMPONENT-PROTOTYPE>
          </COMPONENTS>
        </COMPOSITION-SW-COMPONENT-TYPE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

Example A.3

```
connector FA_Connection {

import_franca  "../franca/componentF.fidl";
import_autosar "../autosar/componentA.arxml"

instances {
  franca_instance g implements franca.F1, franca.F2
```

```
}

connections {
  AF_CS     autosar_port autosar.C : a : autosar.A.reqPort_CS      ->
     franca_instance g : F1
  AF_SRPull autosar_port autosar.C : a : autosar.A.provPort_SRPull ->
     franca_instance g : F2
  AF_SRPush autosar_port autosar.C : a : autosar.A.provPort_SRPush ->
     franca_instance g

  FA_SR franca_instance g : F1 -> autosar_port autosar.C : a : autosar.A.
     reqPort_SR
  FA_CS franca_instance g      -> autosar_port autosar.C : a : autosar.A.
     provPort_CS
}
}
```

The result of the translation of the Franca Connector (A.3) is indicated in listing A.4. Only the composition software component type *C* is shown that contains the two component prototypes *a* and *g*, and the five assembly connectors *AF_CS*, *AF_SRPull*, *AF_SRPush*, *FA_SR*, and *FA_CS*.

Example A.4

```
<COMPOSITION-SW-COMPONENT-TYPE UUID="acb0061a-b365-32d4-b0e4-9fea3427044d">
<SHORT-NAME>C</SHORT-NAME>
  <COMPONENTS>
    <SW-COMPONENT-PROTOTYPE UUID="2dcf4e8f-1ca0-3270-b247-9278f627d8ee">
      <SHORT-NAME>a</SHORT-NAME>
      <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/autosar/A</TYPE-TREF
         >
    </SW-COMPONENT-PROTOTYPE>
    <SW-COMPONENT-PROTOTYPE UUID="269da27d-413f-4969-a931-0fa13019360c">
      <SHORT-NAME>g</SHORT-NAME>
      <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/SwComponentTypes/
         type_g</TYPE-TREF>
    </SW-COMPONENT-PROTOTYPE>
  </COMPONENTS>
  <CONNECTORS>
    <ASSEMBLY-SW-CONNECTOR UUID="b33c857a-681f-44c2-a622-2f97d1548e30">
      <SHORT-NAME>AF_CS</SHORT-NAME>
      <PROVIDER-IREF>
        <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/autosar/C/g</
           CONTEXT-COMPONENT-REF>
        <TARGET-P-PORT-REF DEST="P-PORT-PROTOTYPE">/SwComponentTypes/type_g
           /csPPort_F1</TARGET-P-PORT-REF>
      </PROVIDER-IREF>
      <REQUESTER-IREF>
        <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/autosar/C/a</
           CONTEXT-COMPONENT-REF>
        <TARGET-R-PORT-REF DEST="R-PORT-PROTOTYPE">/autosar/A/reqPort_CS</
           TARGET-R-PORT-REF>
      </REQUESTER-IREF>
    </ASSEMBLY-SW-CONNECTOR>
```

```
<ASSEMBLY-SW-CONNECTOR UUID="6644e6ab-b794-4547-bf01-f6d5b158bb86">
  <SHORT-NAME>AF_SRPull</SHORT-NAME>
  <PROVIDER-IREF>
    <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/autosar/C/a</
        CONTEXT-COMPONENT-REF>
    <TARGET-P-PORT-REF DEST="P-PORT-PROTOTYPE">/autosar/A/
        provPort_SRPull</TARGET-P-PORT-REF>
  </PROVIDER-IREF>
  <REQUESTER-IREF>
    <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/autosar/C/g</
        CONTEXT-COMPONENT-REF>
    <TARGET-R-PORT-REF DEST="R-PORT-PROTOTYPE">/SwComponentTypes/type_g
        /srRPort_F2</TARGET-R-PORT-REF>
  </REQUESTER-IREF>
</ASSEMBLY-SW-CONNECTOR>
<ASSEMBLY-SW-CONNECTOR UUID="cca795d9-1e42-4842-b65b-15805f37d04f">
  <SHORT-NAME>AF_SRPush</SHORT-NAME>
  <PROVIDER-IREF>
    <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/autosar/C/a</
        CONTEXT-COMPONENT-REF>
    <TARGET-P-PORT-REF DEST="P-PORT-PROTOTYPE">/autosar/A/
        provPort_SRPush</TARGET-P-PORT-REF>
  </PROVIDER-IREF>
  <REQUESTER-IREF>
    <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/autosar/C/g</
        CONTEXT-COMPONENT-REF>
    <TARGET-R-PORT-REF DEST="R-PORT-PROTOTYPE">/SwComponentTypes/type_g
        /provPort_SRPush</TARGET-R-PORT-REF>
  </REQUESTER-IREF>
</ASSEMBLY-SW-CONNECTOR>
<ASSEMBLY-SW-CONNECTOR UUID="ce3b9a48-65ec-43bd-af47-61c27615e42c">
  <SHORT-NAME>FA_SR</SHORT-NAME>
  <PROVIDER-IREF>
    <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/autosar/C/g</
        CONTEXT-COMPONENT-REF>
    <TARGET-P-PORT-REF DEST="P-PORT-PROTOTYPE">/SwComponentTypes/type_g
        /srPPort_F1</TARGET-P-PORT-REF>
  </PROVIDER-IREF>
  <REQUESTER-IREF>
    <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/autosar/C/a</
        CONTEXT-COMPONENT-REF>
    <TARGET-R-PORT-REF DEST="R-PORT-PROTOTYPE">/autosar/A/reqPort_SR</
        TARGET-R-PORT-REF>
  </REQUESTER-IREF>
</ASSEMBLY-SW-CONNECTOR>
<ASSEMBLY-SW-CONNECTOR UUID="8d417245-bd8d-4dde-bfd8-ae01370ce907">
  <SHORT-NAME>FA_CS</SHORT-NAME>
  <PROVIDER-IREF>
    <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/autosar/C/a</
        CONTEXT-COMPONENT-REF>
    <TARGET-P-PORT-REF DEST="P-PORT-PROTOTYPE">/autosar/A/provPort_CS</
        TARGET-P-PORT-REF>
  </PROVIDER-IREF>
  <REQUESTER-IREF>
    <CONTEXT-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">/autosar/C/g</
        CONTEXT-COMPONENT-REF>
```

```
        <TARGET-R-PORT-REF DEST="R-PORT-PROTOTYPE">/SwComponentTypes/type_g
            /provPort_CS</TARGET-R-PORT-REF>
      </REQUESTER-IREF>
    </ASSEMBLY-SW-CONNECTOR>
  </CONNECTORS>
</COMPOSITION-SW-COMPONENT-TYPE>
```

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	**ARPackage**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	AUTOSAR package, allowing to create top level packages to structure the contained ARElements. ARPackages are open sets. This means that in a file based description system multiple files can be used to partially describe the contents of a package. This is an extended version of MSR's SW-SYSTEM.			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arPackage	ARPackage	*	aggr	This represents a sub package within an ARPackage, thus allowing for an unlimited package hierarchy. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=arPackage.shortName, arPackage.variation Point.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
element	PackageableElement	*	aggr	Elements that are part of this package **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=element.shortName, element.definition, element.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=20
referenceBase	ReferenceBase	*	aggr	This denotes the reference bases for the package. This is the basis for all relative references within the package. The base needs to be selected according to the base attribute within the references. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=referenceBase.shortLabel xml.sequenceOffset=10

Table B.1: ARPackage

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*, *Atp Blueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dynamicArray SizeProfile	String	0..1	attr	Specifies the profile which the array will follow if it is a variable size array.

▽

△

Class	ApplicationArrayDataType			
element	ApplicationArray Element	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 B.2: ApplicationArrayDataType

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*, ApplicationPrimitiveDataType			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table B.3: ApplicationDataType

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

Table B.4: ApplicationPrimitiveDataType

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*, *Atp Blueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	**ApplicationRecordDataType**			
element (ordered)	ApplicationRecord Element	*	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 ApplicationrecordData Type. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table B.5: ApplicationRecordDataType

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 ApplicationRecord Element 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.6: ApplicationRecordElement

Class	**ApplicationSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ApplicationSwComponentType is used to represent the application software. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table B.7: ApplicationSwComponentType

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*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
direction	ArgumentDirection Enum	0..1	attr	This attribute specifies the direction of the argument prototype.

▽

△

Class	**ArgumentDataPrototype**			
serverArgument ImplPolicy	ServerArgumentImpl PolicyEnum	0..1	attr	This defines how the argument type of the servers RunnableEntity is implemented. If the attribute is not defined this has the same semantics as if the attribute is set to the value useArgumentType for primitive arguments and structures.

Table B.8: ArgumentDataPrototype

Class	**AssemblySwConnector**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	AssemblySwConnectors are exclusively used to connect SwComponentPrototypes in the context of a CompositionSwComponentType.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *SwConnector*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
provider	AbstractProvidedPort Prototype	0..1	iref	Instance of providing port. **InstanceRef implemented by:**PPortInComposition InstanceRef
requester	AbstractRequiredPort Prototype	0..1	iref	Instance of requiring port. **InstanceRef implemented by:**RPortInComposition InstanceRef

Table B.9: AssemblySwConnector

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, EcuAbstractionSwComponent Type, NvBlockSwComponentType, SensorActuatorSwComponentType, ServiceProxySwComponent Type, ServiceSwComponentType			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
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.10: AtomicSwComponentType

Class	**ClientServerInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A client/server interface declares a number of operations that can be invoked on a server by a client. **Tags:**atp.recommendedPackage=PortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
operation	ClientServerOperation	*	aggr	ClientServerOperation(s) of this ClientServerInterface. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=blueprintDerivationTime
possibleError	ApplicationError	*	aggr	Application errors that are defined as part of this interface.

Table B.11: ClientServerInterface

Class	**ClientServerInterfaceMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of ClientServerOperations in context of two different ClientServerInterfaces.			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *Identifiable*, *MultilanguageReferrable*, *PortInterfaceMapping*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
errorMapping	ClientServerApplication ErrorMapping	*	aggr	Map two different ApplicationErrors defined in the context of two different ClientServerInterfaces.
operation Mapping	ClientServerOperation Mapping	*	aggr	Mapping of two ClientServerOperations in two different ClientServerInterfaces

Table B.12: ClientServerInterfaceMapping

Class	**ClientServerOperation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An operation declared within the scope of a client/server 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
diagArgIntegrity	Boolean	0..1	attr	This attribute shall only be used in the implementation of diagnostic routines to support the case where input and output arguments are allocated in a shared buffer and might unintentionally overwrite input arguments by tentative write operations to output arguments. This situation can happen during sliced execution or while output parameters are arrays (call by reference). The value true means that the ClientServerOperation is aware of the usage of a shared buffer and takes precautions to avoid unintentional overwrite of input arguments. If the attribute does not exist or is set to false the Client ServerOperation does not have to consider the usage of a shared buffer.
possibleError	ApplicationError	*	ref	Possible errors that may by raised by the referring operation.

Table B.13: ClientServerOperation

Class	**CompositionSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	A CompositionSwComponentType aggregates SwComponentPrototypes (that in turn are typed by Sw ComponentTypes) 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	SwComponent Prototype	*	aggr	The instantiated components that are part of this composition. The aggregation of SwComponentPrototype is subject to variability with the purpose to support the conditional existence of a SwComponentPrototype. Please be aware: if the conditional existence of Sw ComponentPrototypes is resolved post-build the deselected SwComponentPrototypes are still contained in the ECUs build but the instances are inactive in that they are not scheduled by the RTE. The aggregation is marked as atpSplitable in order to allow the addition of service components to the ECU extract during the ECU integration. The use case for having 0 components owned by the CompositionSwComponentType could be to deliver an empty CompositionSwComponentType to e.g. a supplier for filling the internal structure. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=component.shortName, component.variation Point.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. The aggregation is marked as atpSplitable in order to allow the extension of the ECU extract with AssemblySw Connectors between ApplicationSwComponentTypes and ServiceSwComponentTypes during the ECU integration. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=connector.shortName, connector.variation Point.shortLabel vh.latestBindingTime=postBuild
constantValue Mapping	ConstantSpecification MappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for initValues of PPortComSpecs and RPortCom Spec. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=constantValueMapping
dataType Mapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the used ApplicationDataTypes in PortInterfaces. Background: when developing subsystems it may happen that ApplicationDataTypes are used on the surface of CompositionSwComponentTypes. In this case it would be ▽

▽

△

Class	CompositionSwComponentType			
				△ reasonable to be able to also provide the intended mapping to the ImplementationDataTypes. However, this mapping shall be informal and not technically binding for the implementors mainly because the RTE generator is not concerned about the CompositionSwComponent Types. Rationale: if the mapping of ApplicationDataTypes on the delegated and inner PortPrototype matches then the mapping to ImplementationDataTypes is not impacting compatibility. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=dataTypeMapping
instantiation RTEEventProps	InstantiationRTEEvent Props	*	aggr	This allows to define instantiation specific properties for RTE Events, in particular for instance specific scheduling. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=instantiationRTEEventProps.shortLabel, instantiationRTEEventProps.variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime

Table B.14: CompositionSwComponentType

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

Table B.15: CompuMethod

Class	**DataTypeMap**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	This class represents the relationship between ApplicationDataType and its implementing Abstract ImplementationDataType.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
applicationDataType	ApplicationDataType	0..1	ref	This is the corresponding ApplicationDataType
implementationDataType	AbstractImplementationDataType	0..1	ref	This is the corresponding AbstractImplementationDataType.

Table B.16: DataTypeMap

Class	**DataTypeMappingSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	This class represents a list of mappings between ApplicationDataTypes and ImplementationDataTypes. In addition, it can contain mappings between ImplementationDataTypes and ModeDeclarationGroups. **Tags:**atp.recommendedPackage=DataTypeMappingSets			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataTypeMap	DataTypeMap	*	aggr	This is one particular association between an ApplicationDataType and its AbstractImplementationDataType.
modeRequestTypeMap	ModeRequestTypeMap	*	aggr	This is one particular association between an ModeDeclarationGroup and its AbstractImplementationDataType.

Table B.17: DataTypeMappingSet

Class	**DelegationSwConnector**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	A delegation connector delegates one inner PortPrototype (a port of a component that is used inside the composition) to a outer PortPrototype of compatible type that belongs directly to the composition (a port that is owned by the composition).			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *SwConnector*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
innerPort	PortPrototype	0..1	iref	The port that belongs to the ComponentPrototype in the composition **Tags:**xml.typeElement=true **InstanceRef implemented by:**PortInCompositionTypeInstanceRef
outerPort	PortPrototype	0..1	ref	The port that is located on the outside of the CompositionType

Table B.18: DelegationSwConnector

Class	**ImplementationDataType**
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes

▽

△

Class	**ImplementationDataType**			
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*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dynamicArray SizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.
isStructWith Optional Element	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)	ImplementationData TypeElement	*	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 Implementation DataType representing a structure. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the Implementation DataType. **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.19: ImplementationDataType

Class	**ImplementationDataTypeElement**			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated. This element either consists of further subElements or it is further defined via its swDataDefProps. There are several use cases within the system of ImplementationDataTypes fur such a local declaration: • It can represent the elements of an array, defining the element type and array size • It can represent an element of a struct, defining its type • It can be the local declaration of a debug element.			
Base	*ARObject*, *AbstractImplementationDataTypeElement*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arrayImplPolicy	ArrayImplPolicyEnum	0..1	attr	This attribute controls the implementation of the payload of an array. It shall only be used if the enclosing ImplementationDataType constitutes an array.

▽

△

Class	**ImplementationDataTypeElement**			
arraySize	PositiveInteger	0..1	attr	The existence of this attributes (if bigger than 0) defines the size of an array and declares that this Implementation DataTypeElement represents the type of each single array element. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
arraySize Handling	ArraySizeHandling Enum	0..1	attr	The way how the size of the array is handled in case of a variable size array.
arraySize Semantics	ArraySizeSemantics Enum	0..1	attr	This attribute controls the meaning of the value of the array size.
isOptional	Boolean	0..1	attr	This attribute represents the ability to declare the enclosing ImplementationDataTypeElement as optional. This means that, at runtime, the ImplementationDataType Element may or may not have a valid value and shall therefore be ignored. The underlying runtime software provides means to set the CppImplementationDataTypeElement as not valid at the sending end of a communication and determine its validity at the receiving end.
subElement (ordered)	ImplementationData TypeElement	*	aggr	Element of an array, struct, or union in case of a nested declaration (i.e. without using "typedefs"). The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a Implementation DataType representing a structure. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
swDataDef Props	SwDataDefProps	0..1	aggr	The properties of this ImplementationDataTypeElement.

Table B.20: ImplementationDataTypeElement

Class	**PPortInCompositionInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*, *PortInCompositionTypeInstanceRef*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
context Component	SwComponent Prototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetPPort	AbstractProvidedPort Prototype	0..1	ref	**Tags:**xml.sequenceOffset=30

Table B.21: PPortInCompositionInstanceRef

Class	**PPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port providing a certain port interface.			
Base	*ARObject*, *AbstractProvidedPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	**PPortPrototype**			
provided Interface	PortInterface	0..1	tref	The interface that this port provides. **Stereotypes:** isOfType

Table B.22: PPortPrototype

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***
clientServer Annotation	ClientServerAnnotation	*	aggr	Annotation of this PortPrototype with respect to client/ server communication.
delegatedPort Annotation	DelegatedPort Annotation	0..1	aggr	Annotations on this delegated port.
ioHwAbstraction Server Annotation	IoHwAbstractionServer Annotation	*	aggr	Annotations on this IO Hardware Abstraction port.
logAndTrace Message CollectionSet	LogAndTraceMessage CollectionSet	0..1	ref	Reference to a collection of Log or Trace messages that will be used by the application. **Tags:**atp.Status=draft
modePort Annotation	ModePortAnnotation	*	aggr	Annotations on this mode port.
nvDataPort Annotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port.
parameterPort Annotation	ParameterPort Annotation	*	aggr	Annotations on this parameter port.
senderReceiver Annotation	SenderReceiver Annotation	*	aggr	Collection of annotations of this ports sender/receiver communication.
triggerPort Annotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port.

Table B.23: PortPrototype

Class	**RPortInCompositionInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*, *PortInCompositionTypeInstanceRef*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
context Component	SwComponent Prototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetRPort	AbstractRequiredPort Prototype	0..1	ref	**Tags:**xml.sequenceOffset=30

Table B.24: RPortInCompositionInstanceRef

Class	**RPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port requiring a certain port interface.			
Base	*ARObject*, *AbstractRequiredPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
mayBe Unconnected	Boolean	0..1	attr	If set to true, this attribute indicates that the enclosing RPortPrototype may be left unconnected and that this aspect has explicitly been considered in the software-component's design.
required Interface	PortInterface	0..1	tref	The interface that this port requires. **Stereotypes:** isOfType

Table B.25: RPortPrototype

Class	**SenderReceiverInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A sender/receiver interface declares a number of data elements to be sent and received. **Tags:**atp.recommendedPackage=PortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DataInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataElement	VariableDataPrototype	*	aggr	The data elements of this SenderReceiverInterface.
invalidation Policy	InvalidationPolicy	*	aggr	InvalidationPolicy for a particular dataElement
metaDataItem Set	MetaDataItemSet	*	aggr	This aggregation defines fixed sets of meta-data items associated with dataElements of the enclosing Sender ReceiverInterface

Table B.26: SenderReceiverInterface

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

Table B.27: 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)			
consistency Needs	ConsistencyNeeds	*	aggr	This represents the collection of ConsistencyNeeds owned by the enclosing SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=consistencyNeeds.shortName, consistency Needs.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
port	PortPrototype	*	aggr	The PortPrototypes through which this SwComponent Type 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.short Label vh.latestBindingTime=preCompileTime
portGroup	PortGroup	*	aggr	A port group being part of this component. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
swcMapping Constraint	SwComponentMapping Constraints	*	ref	Reference to constraints that are valid for this Sw ComponentType.
swComponent Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=swComponentDocumentation, sw ComponentDocumentation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=-10
unitGroup	UnitGroup	*	ref	This allows for the specification of which UnitGroups are relevant in the context of referencing SwComponentType.

Table B.28: SwComponentType

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

Table B.29: VariableDataPrototype