Document Title	Specification of System Template
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Version	2.0.0
Document Status	Draft
Part of Release	2.1
Revision	0014

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
31.01.2007	2.0.0	AUTOSAR Administration	• Support for Signal Groups added. • Rework of the Topology Description • Introduction of PDUs. Description of the PDU Multiplexer, PDU Gateway. • FlexRay: multiple transmission of a frame within one communication cycle is supported now. • Removed the concept of Variant Descriptions (Properties) and CompToECUMappingConstraints relying on the property concept. • Split SwCompToEcuMapping in two classes in order to allow separation of SWC-to-ECU mapping and Implementation-to-SWC mapping. • Removed preliminary chapter on MOST as it is not part of the standard. • For all Instance References in the System Template added diagrams to the meta-model containing detailed representations of these references. • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
09.05.05	1.0.0	AUTOSAR Administration	Initial release.

Release Notes

Errata and known deficiencies

This chapter describes which classes of the System Template are released and which classes are not yet finalized. The classes that are not yet finalized will likely be changed, to some extent, in a future release of the System Template.

Released classes

Classes that are released are listed in the table below.

Class	***Description***	***Defined in chapter***
System	System is the main class that aggregates all information relevant to describe a system and its constraints.	3.1
SoftwareComposition	This top-level software composition contains the functionality of the full system and describes the complete application software architecture of this system.	1.3[1]
SwCompToEcuMapping	With the SwCompToEcuMapping element it is possible to express the mapping of Software components to one ECU instance.	6.1.1
ECUInstance	The software configurable properties of the ECU are described in the ECUInstance.	4.7
SystemTopologyType	The SystemTopologyType may be reused in different systems.	4.2
SystemTopologyInstance	The topology of the system. Since several systems may share the same topology, a reference to a SystemTopologyType is used to define which topology is used in the system.	4.2
TopologyElement	An abstract class that is a generalization of ECUInstance and Hub.	4.2
SenderReceiverToSignalMapping	Mapping of a sender receiver communication data element with a primitive data type to a signal.	6.2.1.1
SenderReceiverToSignalGroupMapping	Mapping of a sender receiver communication data element with a composite data type to a signal group.	6.2.1.2
SenderRecRecordTypeMapping	Used if the compositeType is a	6.2.1.2

[1] Defined in SW-C Template, used in the context of System Template.

	Record.	
SenderRecRecordElementMapping	Mapping of a primitive record element to a SystemSignal.	6.2.1.2
SenderRecCompositeTypeMapping	Abstract class	6.2.1.2
SenderRecArrayTypeMapping	Used if the compositeType is an Array.	6.2.1.2
SenderRecArrayElementMapping	The ArrayElement may be a primitive one or a composite one. A primitive array element is mapped to a SystemSignal.	6.2.1.2
ClientServerToSignalGroupMapping	Mapping of client server operation arguments to signals of a signal group.	6.2.1.3
ClientId	In case of a server on one ECU with multiple clients on other ECUs, the client server communication shall use different unique COM signals and signal groups for each client to allow the identification of the client associated with each system signal.	6.2.1.3
ApplicationError	This is a user-defined error that is associated with an element of an AUTOSAR interface.	6.2.1.3
ClientServerPrimitiveTypeMapping	Mapping of an argument with a primitive data type to a signal.	6.2.1.3
SequenceCounter	The purpose of sequence counters is to map a response to the correct request of a known client.	6.2.1.3
EmptySignal	According to the COM Specification, signal groups without signals are allowed.	6.2.1.3
ClientServerCompositeTypeMapping	An abstract class	6.2.1.3
ClientServerArrayTypeMapping	Used if the compositeType is an array.	6.2.1.3
ClientServerArrayElementMapping	The ArrayElement may be a primitive one or a composite one. If the element is primitive, it will be mapped to a SystemSignal.	6.2.1.3
ClientServerRecordTypeMapping	Used if the compositeType is a Record.	6.2.1.3
ClientServerRecordElementMapping	Mapping of a primitive record element to a SystemSignal.	6.2.1.3
SystemSignal	The system signal represents the communication system's view of data exchanged between SW components which reside on different ECUs.	5.1
SystemSignalGroup	A signal group refers to a set of signals that must always be kept together. A signal group is used to guarantee the atomic transfer of AUTOSAR composite data types.	5.3

Classes aggregated by a released class are not automatically released, if not explicit defined as released classes.

Non-released classes

All classes in System Template not listed in the table are not released and will likely be changed, at some extent, in a future release of the System Template. Backwards compatibility can not be assumed for not released classes.

Disclaimer

Any use of these specifications requires membership within the AUTOSAR Development Partnership or an agreement with the AUTOSAR Development Partnership. The AUTOSAR Development Partnership will not be liable for any use of these specifications.

Following the completion of the development of the AUTOSAR specifications commercial exploitation licenses will be made available to end users by way of written License Agreement only.

Advice to users of AUTOSAR Specification Documents:

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).
Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction

1.1 Methodology for defining formal template

Figure 1 illustrates the overall methodology used to define formal templates. As is explained in the "*Template UML Profile and Modeling Guide*", it is important to separate a precise and concise model of the information that needs to be captured from the concrete XML-DTDs, XML-Schemas or other technology that is used to define the actual templates.

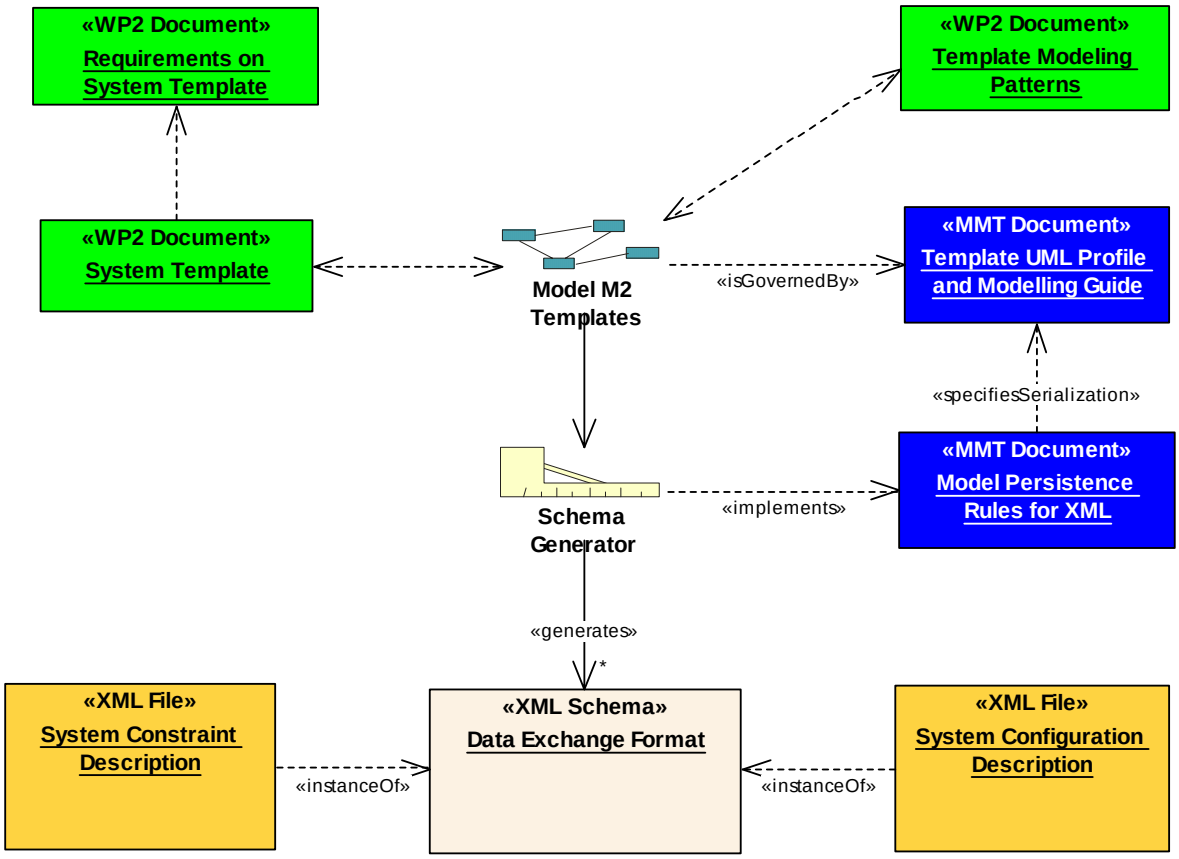

Figure 1: Methodology to define templates in AUTOSAR

The following documents describe the various aspects of the methodology:
The document called "*System Template*" (this document) describes the information that can be captured in the "system constraint" and "system configuration" description, independently from the mapping of this model on XML-technology. This document consists of the metamodel and an elaborate description of the semantics (the precise meaning) of all the information that can be captured with this metamodel
The document called "*Template UML Profile and Modeling Guide*" describes the basic concepts that should be used when modeling the information in the metamodel.
The document called "*Model Persistence Rules for XML*" describes how XML is used and how the metamodel designed in the "System Template" should be translated by the "*Schema Generator*" (MDS) into XML-Schema (XSD) "*Data Exchange Format*". This "formalization strategy" is to be used for all data that is formally described in the

metamodel. In particular this document is worth to read in order to understand the mapping of the metamodel and the XML based System template.
The "*Template Modeling Patterns*" are represented as predefined Classes in the metamodel which are incorporated in the generated schema. Examples for such patterns are the "common attributes" which are added to each generated class even if not explicitly inherited in the metamodel.
The concrete "Template" the "*Data Exchange Format*" is an XML schema which is generated out of the metamodel described in the "*System Template*" using the approach and the patterns defined in the "*Model Persistence Rules for XML*". This schema is typically used as input to tools. The M1 – level system descriptions are XML files which can be validated against the schema. In that sense they are instances of the schema defining the XML representation of the template.

1.2 Scope

This document describes the system template and its use for the System Constraint Description and the System Configuration Description.
In general a filled system template defines the relationship between the pure Software View on the System (represented by a top level SW Component Composition) and a Physical System Architecture with networked ECU instances. The system template is used in two stages of the AUTOSAR Methodology (see **Figure 2**).

- As System Constraint Description it serves as input to the AUTOSAR system generator
- As System Configuration Description, it defines the output of the AUTOSAR System Configuration Generator and serves as input to the AUTOSAR ECU Configuration Generator for the different ECUs defined in the description.

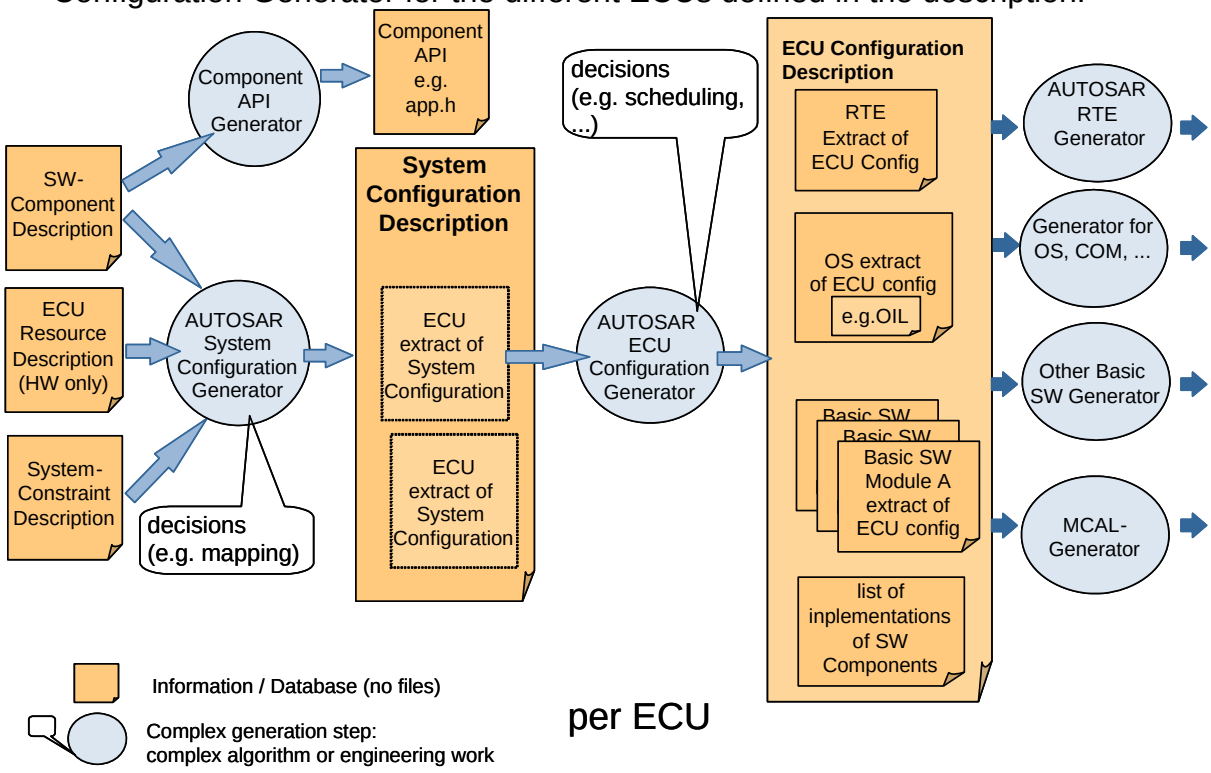

Figure 2: AUTOSAR Methodology

The System Template defines five major elements: Topology, Software, Communication, Mapping and Mapping Constraints, which will be defined in detail in the following chapters. **Figure 3** gives an overview how these are used in the two different descriptions.

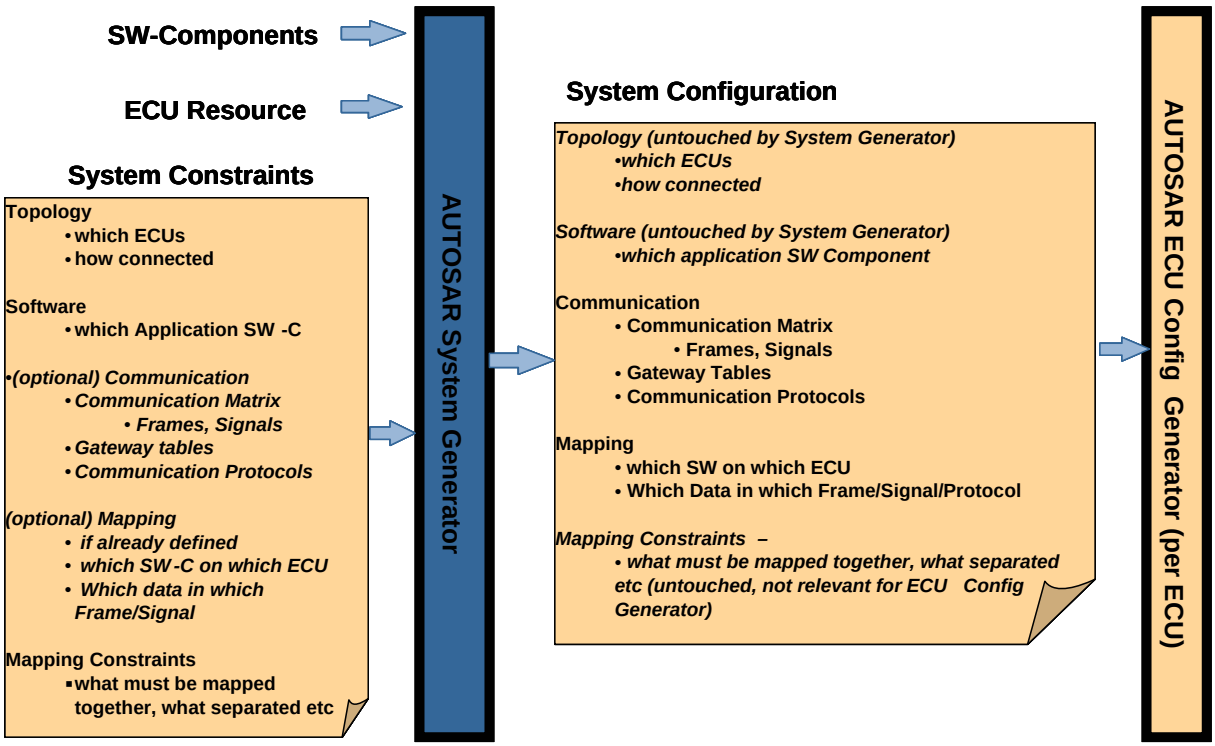

Figure 3: Scope of System Constraint Description and System Configuration Description

On **Figure 3** some of the elements are marked *optional* for the System Constraint Description. If one starts with a new AUTOSAR project, these elements may not be present in the System Constraint Description. No (at least partial) functionality has been mapped yet, thus the communication matrix is not populated. But in most cases, many functional mappings are already predefined and contribute to the population of the communication matrix with their associated signals, thus being present in the System Constraint Description.

Reasons for such a predefinition are manifold. In some cases, hardware setup dictates where certain functionality resides, in some cases, a partial or complete communication matrix and/or completely configured ECUs (HW and SW) of another system (vehicle) has to be taken over. This approach is eased by the fact that System Configuration and System Constraint Description use the same format. That way it is possible to reuse parts of a System Configuration Description of the other system/vehicle in the actual System Constraint Description.

Furthermore, in the figure some of the elements are marked *untouched* for the System Configuration Description. This can have two reasons:

- the System Generator does not modify neither the Topology (networked ECUs) nor the Software, so these parts are just moved from System Constraint Description to System Configuration Description during the generation step.
- In a completed System Configuration Description, all SW components and all ECU-to-ECU communication have been mapped. Thus mapping constraints that

limit the flexibility in the mapping phase of the system generator are obsolete and will not be used in subsequent generator steps. They may however still be present for documentation and validation reasons.

Even if the communication matrix is determined as the result of the system configuration, the ECUs still have to be configured. This is done by the ECU configuration generator, which takes the System Configuration description as input and generates the ECU configuration description. The following guiding principles have been used to determine which information must be part of the System Configuration Description and which goes into the ECU Configuration Description:

- information that is common for several ECUs and has to be agreed, must be part of the System Configuration Description and is thus covered by the System Template.
- information, that only has ECU-local relevance is part of the ECU Configuration Description.

Thus the ECU Configuration Description will include the OS-schedule, the RTE-configuration and last but not least the configuration of the ECU basic software including the concrete communication drivers on that ECU.
Wherever applicable concepts from FIBEX [13] are used. See chapter 9.2.1.

1.3 Document Overview

Figure 4 shows the separation between the ECU resource description, the SW-component description and the System Template (System Constraint Description, System Configuration Description). This Specification of the System Template consists of three main parts `Mapping`, `Topology` and `Communication`. These areas form also the main chapters in this specification. The individual elements within these areas will be explained in more detail in the following chapters.

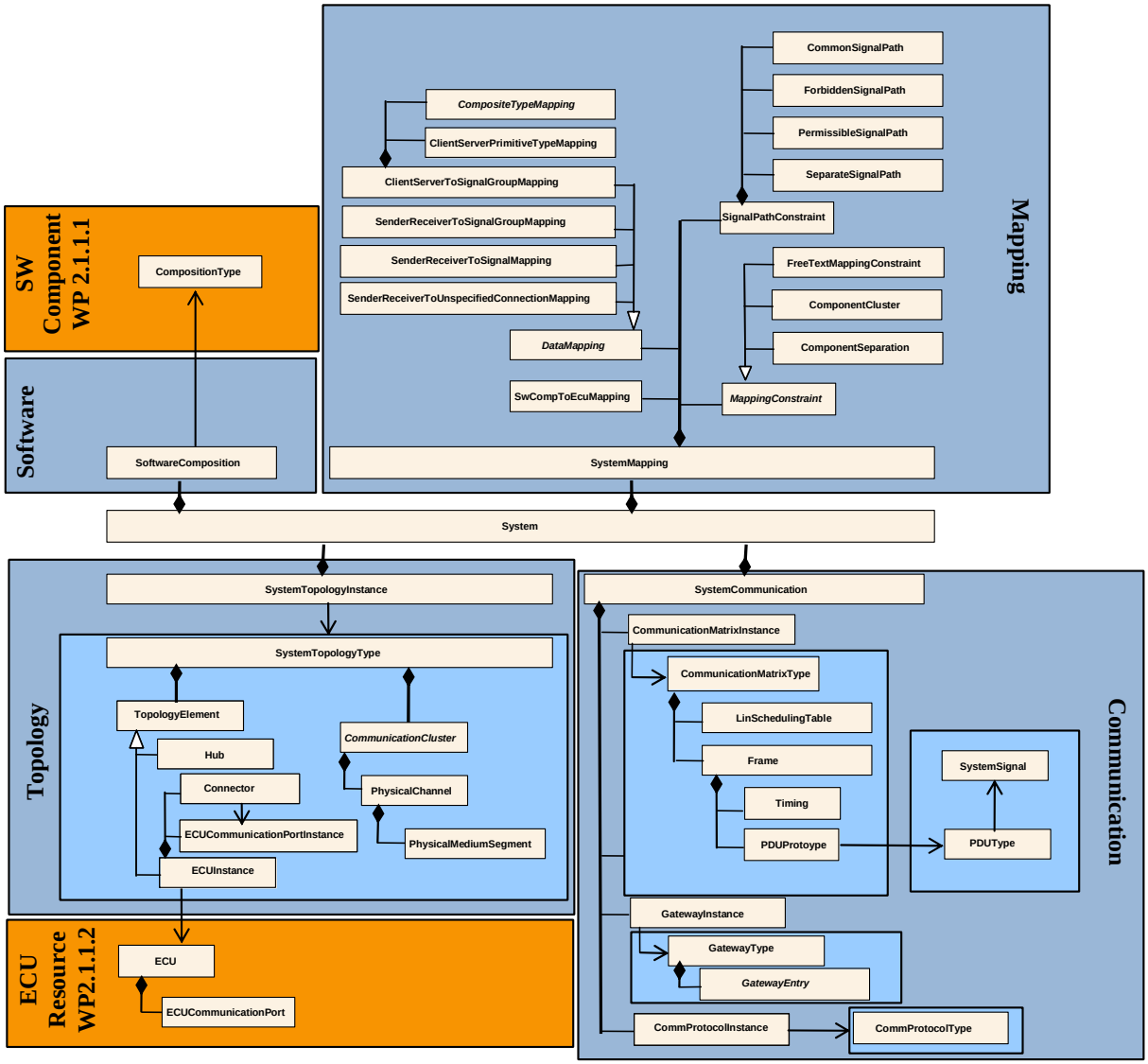

Figure 4: Overview about the System Template

The ECU Resource Template deals with the description of the hardware resources of an ECU. The collection of all ECUs, which are integrated in the car, are described in the topology part of the System Configuration Description/System Constraint Description. Each of these `ECUInstances` uses the ECU Resource Template to describe the hardware resources. That's the reason, why the topology part has a reference to the ECU Resource Description. Furthermore, also the communication links between the `ECUInstances` are described in the topology part.

The data frames that will be transferred via these communication links are described in the communication part. Also the gateway tables, which define the copying of frames or signals from one bus to another bus, are covered in this part.

The SW component description describes the SW components as well as their communication by logical signals. The top-level software composition is part of the System Template (Software). This top-level software composition contains the functionality of the full system and describes the complete application software architecture of this system.

The definition of this SoftwareComposition uses the elements defined in the SW Component Template, like e.g. ComponentType, PortInterface, Assembly-ConnectorPrototype and DelegationConnectorPrototype. Thus, the area Software contains a reference to the Software Component Description. The top level software composition is described in more detail in [9].

The area Mapping deals with mapping of software components to ECUs as well as the mapping of data elements that are to be exchanged between software components according to the AssemblyConnectorPrototypes in the Software-Composition onto frames or communication protocols.

It is possible, that certain Information about the System is covered implicit by more than one entry in the System Description:

- To hold the description flexible (e.g. for future extensions)
- To make it possible to describe the system generator input (System Constraint Description) and the output (System Configuration Description) with the same Template.

Everywhere such an over determined description is possible, the affected parameters have to be verified for consistency, which is always achieved automatically by a tool.

2 Requirements Traceability

Requirement	***Satisfied By***
SysCT1 Mixed Systems (AUTOSAR/NON-AUTOSAR)	5.4 Communication Matrix
SysCT2 Basic Software Resources and RTE Resources	Chapter 6.3 RTE and basic software resource estimations
SysCT3 Iterative Development	Chapter 1.2 Scope
SysCT4 Variant handling	Not yet covered.
SysCT5 Timing requirements	Not yet covered.
SysCT6 Compatibility between the WP2.1.1.x Templates	Is covered by a common meta-model: Chapter 3.1 Structure of the overall model
SysCT7 Mapping of Software Components to ECUs	Chapter 6.1 Software component mapping
SysCT8 SWC Cluster	Chapter 6.1.3 Software component mapping constraints
SysCT9 SWC Separation	Chapter 6.1.3 Software component mapping constraints
SysCT10 Exclusive Mapping of SW-C	Not yet covered.
SysCT11 Dedicated Mapping of SW-C	Not yet covered.
SysCT13 Topology	Chapter 4 Topology
SysCT14 Data Segmenting	Chapter 5.6 Communication protocols
SysCT15 Bus bandwidth	Chapter4 Topology; Chapter 5.4.1.2 Timing
SysCT16 Dedicated physical connections	Chapter 6.2.2 Communication Path
SysCT17 Mapping of signals to the same physical line	Chapter 6.2.2 Communication Path
SysCT18 Mapping of signals to different physical lines	Chapter 6.2.2 Communication Path
SysCT19 Mapping of signals to a specific physical line	Chapter 6.2.2 Communication Path
SysCT20 Exclusion of signals from a specific physical line	Chapter 6.2.2 Communication Path
SysCT21 ECU Communication via CAN	Chapter 4.4.1 CANBus (Topology) Chapter 5.4.2.1 CAN (Communication Matrix)
SysCT22 ECU Communication via LIN	Chapter 4.4.3 LINSubBus (Topology) Chapter 5.4.2.3 LIN (Communication Matrix)
SysCT23 ECU Communication via MOST	Not yet covered.
SysCT24 ECU Communication via FlexRay	Chapter 4.4.2 FlexRayCluster (Topology) Chapter 5.4.2.2 FlexRay (Com. Matrix)

3 UML Meta-Model

This chapter gives an overview of the System Template's Unified Modeling Language (UML) meta-model.
All AUTOSAR templates use a common meta-model. Currently three templates are developed in parallel, describing (a) software components, (b) ECU resources and (c) the System Template.
The System Template defines all elements, their parameters and their relations, which are necessary for the System Constraint Description and the System Configuration Description. The meta-model follows the AUTOSAR Modeling guide [3].
This modeling guide explains the exact semantics of the UML elements for AUTOSAR and is mandatory precondition to understand the meta-model. This chapter gives a top-level overview of the meta-model; later chapters refine those concepts and structures. The meta-model will be the input for formal templates such as XML-DTDs (Extensible Markup Language - Document Type Definition) or XML-Schemas. The "Model Persistence Rules for XML" document describes how a W3C XML schema specification compliant XML schema can be compiled out of the AUTOSAR metamodel [4].

3.1 Structure of the overall Model

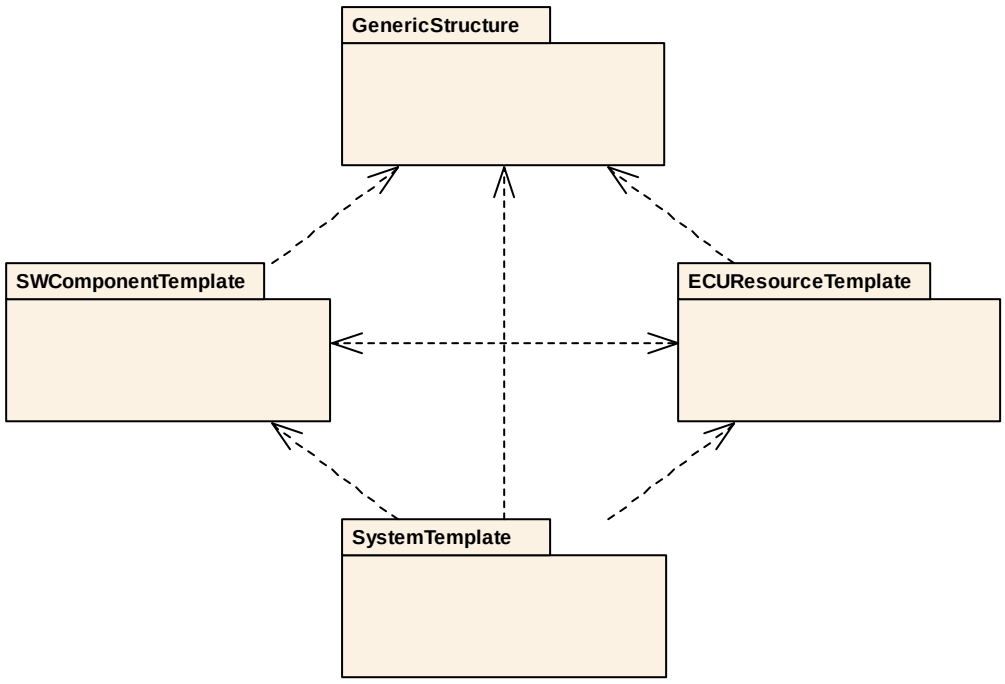

Figure 5: Package Overview

Figure 5 shows the overall structure of the meta-model. As mentioned already, references exist between the `SystemTemplate`, `SWComponentTemplate` and `ECUResourceTemplate`. The package `Generic Structure` contains template independent definitions, e.g. the fact that template elements have unique identifiers.

Furthermore, all templates need to follow the generic structure introduced in this chapter.

Every template starts with an element AUTOSAR. While the models created in accordance to this guide are independent of the used formalization, it may still help the reader's understanding to note that AUTOSAR would also typically be the root element of a XML Schema generated from such a model.

AUTOSAR can then contain one or more nested packages, simply allowing to further structure the contents of the M1 model[2]. Several ARElements are defined on top level for System Template exclusive use, as shown in the following

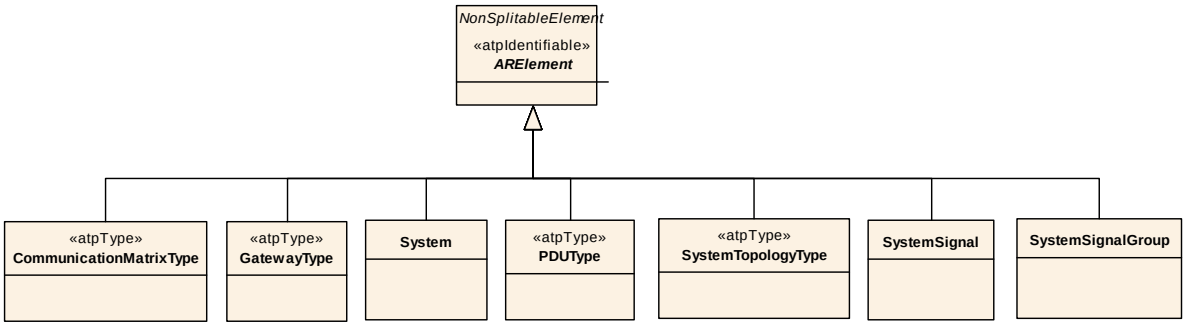

Figure 6: ARElements in the System Template (SysCtArElement)

- CommunicationMatrixType allows to specify type definitions for communication matrices. A communication matrix defines frames that are transported on a specific communication system, with timing etc. A communication matrix described using the CommunicationMatrixType can be re-used in several System Descriptions to define communication matrix instances. This element is described in detail in chapter 5.4.
- GatewayType allows to specify Gateway tables. These types can be re-used in several System Descriptions to define Gateway instances. This element is described in detail in chapter 5.5.
- System is the main class that aggregates all information relevant to describe a system and its constraints. This element is introduced in the following chapter and refined further on.
- CommProtocolType allows to specify type definitions for communication protocols. These types can be used in the System Description to define communication protocol instances. This element is described in detail in chapter 5.6.

[2] A model and its meta-model are said to be on different meta levels (also referred to as abstraction levels). In AUTOSAR a five layer meta-model hierarchy is used, consisting of the five meta levels M0, M1, M2, M3 and M4 where entities in M0 are expressed in terms of M1 entities, M1 is expressed in terms of M2 entities and so on. The AUTOSAR meta-model hierarchy is described in more detail in the AUTOSAR Modeling guide [5].

- PDUType allows to specify type definitions for PDUs (Protocol Data Units). These types can be re-used in one or several System Descriptions to define PDUs that are transported in a frame on a specific bus. This element is described in detail in chapter 5.4.1.3.
- SystemTopologyType is used to define topologies with ECUs and communication systems connecting them. Such a topology can be re-used in several System Descriptions to define system topology instances. This element is described in detail in chapter 4.
- SystemSignal allows frame independent definition of inter-ECU communication. This element is described in detail in chapter 5.1.
- SystemSignalGroup allows to group SystemSignals that must always be kept together in a common frame. This element is described in detail in chapter 5.1.

3.2 Top Level View- System Description

The Top Level View as shown on **Figure 7** gives an overview of the topics, which are treated in the System Template.
The topology part of the System Template allows to define all ECUs of a system and all communication links that interconnect these ECUs. These ECUs are instances of an ECU Description, so every ECUInstance in the System Description contains a link to an ECU (type) defined using the ECU resource template. The system topology can be reused in different systems. Thus, several topology instances can reference to the same system topology type.
The communication cluster is the main element to describe all kinds of communication systems. Each communication cluster is composed of one or more physical channels. The ECUInstances are connected with each other via physical channels (not shown in **Figure 7**).
The definition of frames that are sent on a certain physical channel is done as part of the communication matrix description. The Communication Matrix can also be reused in different systems. Thus, several CommunicationMatrixInstances (normally from different systems) can reference to the same communicationMatrixType.
The communication matrix is separated from the topology, since the same topology may be used with different communication matrices. As a consequence, there may be several communication matrices referencing to the same communication cluster. But in each actual system, there is a 1-to-1 relation between communication clusters and communication matrices.
Reuse of a communication matrix in several systems is however only possible if these systems use the same topology (i.e. system topology type) as well.
Also gateway tables can be defined there, which contain information how frames and signals on one communication system are to be forwarded on another communication system.
The same GatewayType may be reused in different systems. This is however only possible if the system topology and communication matrices that are referenced from the gateway are reused as well.
The Gateway is aggregated directly by the System element and has a reference to one ECUInstance that is the ECU that will implement this gateway. The Gateways are described in more detail in chapter 5.5.

Finally, communication protocol instances can be defined that can be used for interaction between software on different ECUs. A transport communication protocol uses several frames or signals exchanged between ECUs to implement complex interaction behavior that cannot be implemented with a single frame or signal.
In order to obtain an interconnection to the software architecture, the System Template references to one top level software composition. This top-level software composition contains the functionality of the full system and describes the complete application software architecture of this system.
Thus there are different kinds of references to the ECU Resource Template and to the Software Component Template. For SW components, a reference to a very complex structure (a composition of SW components) exists. This is a contrast to the ECU element in the `ECUResourceTemplate`, which describes only individual ECUs, no collection which ECUs are part of the car. Only the System Template describes this technical composition by means of the topology.
Furthermore, all restrictions regarding the mapping of the SW components onto the ECUs and the mapping of signals onto frames are described by the `SystemMapping` meta class.

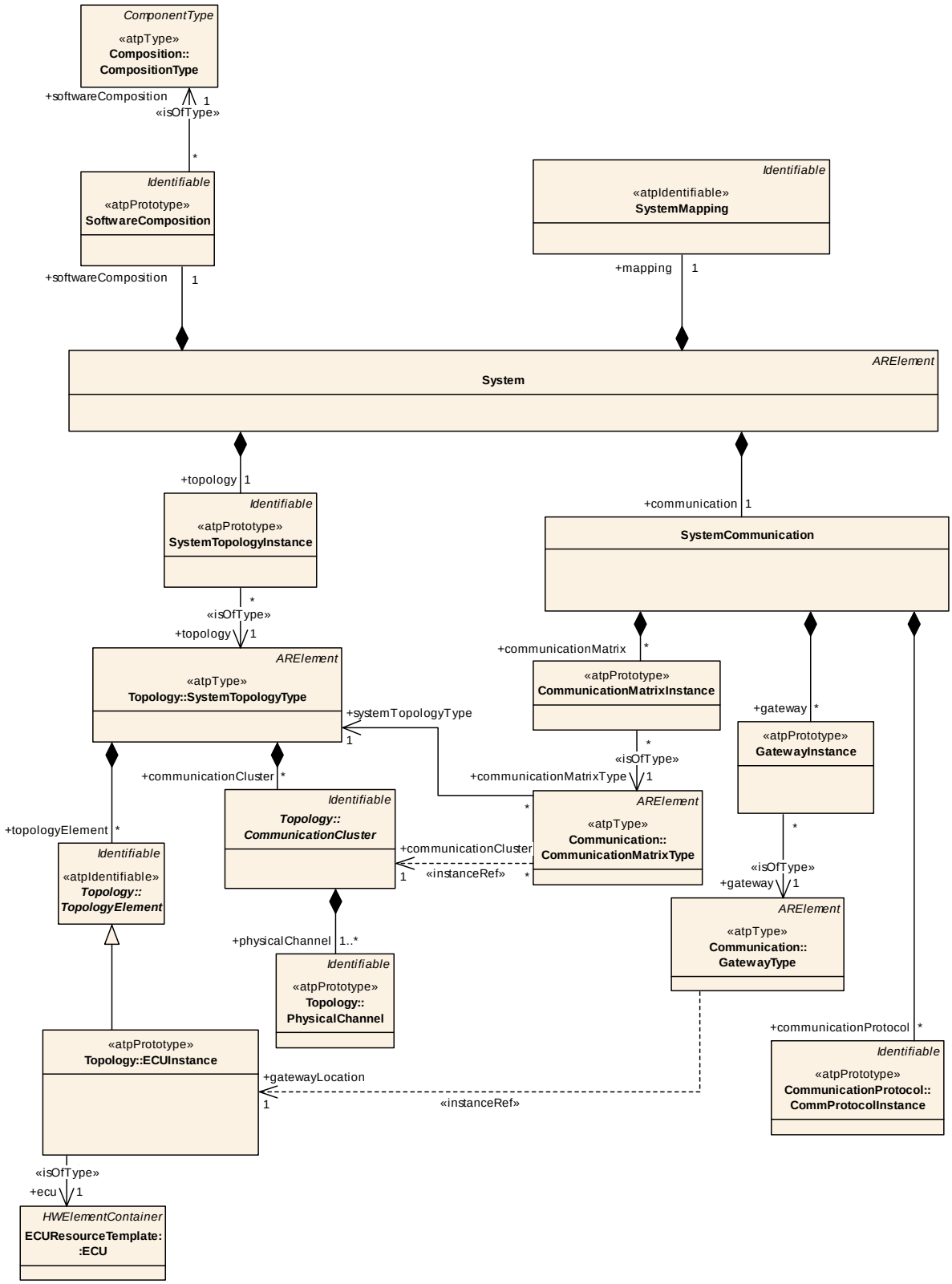

Figure 7: Top Level View (System Template)

3.3 Topology Description

Within the SystemTopologyType, compare overview on **Figure 9**, the meta class CommunicationCluster is the main element to describe the topological connection of communicating ECUs. It aggregates one or more physical channels. A PhysicalChannel represents the communication medium.
To describe all possible network topologies (bus, star, ring, tree) the PhysicalChannel class aggregates 1..n physical medium segment descriptions. Each physical medium segment connects ECUs with each other. If a physical channel is implemented by several physical medium segments, these are connected with each other via (active or passive) hubs. Thus, the PhysicalMediumSegment has references to the ECUs and to the hubs, which it is connected to.
The following picture shows an example. The black circles represent the hubs, and the lines between the hubs and ECUs are the physical medium segments.

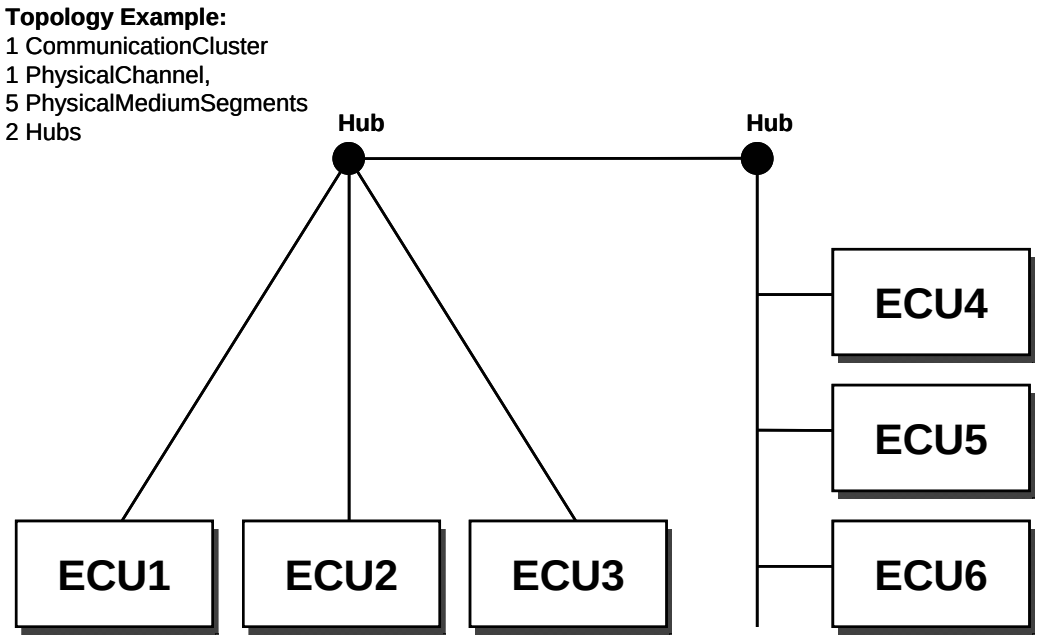

Figure 8: A Topology Example with hubs and physical segments

In the example, a single physical channel is shown, which consists of five segments. ECUs 1 to 3 are connected to a hub via individual segments (as sometimes used in FlexRay), whereas ECUs 4 to 6 share a single segment (As typically used in CAN or LIN).
In a ring topology the position of an ECU relative to other ECUs in the ring can be important. Therefore the physical order of the ECUs is described through the UML constraint {ordered} at the reference between the PhysicalMediumSegment class and the Connector class. The connector class is used to describe the bus interfaces of the ECU.
A tree topology is a collection of star and bus networks arranged in a hierarchy. Each TopologyElement (Hub and ECUInstance) can contain additional parent and child relations to other nodes in the network. These relations can be used to describe the hierarchy in the tree.

The ECUCommunicationPorts are described by WP2.1.1.2. The ECUCommunicationPortInstance class refers to ECUCommunicationPort class and has some own attributes. The attributes may partly be the same or similar to those defined in the ECU resource template. But it is important to repeat those attributes since the hardware might support different values (specified in the ECUCommunicationPort class) than those supported by basic software (specified in the ECUCommunicationPortInstance class). Thus the values specified in the ECUCommunicationPortInstance must always be within the range specified in the related ECUCommunicationPort element.
Examples for such behavior are:

- The ECU hardware may support wakeup on a specific bus (since the transceivers support it), but basic software (drivers) might not implement support for this wakeup, or the ECU mapper has decided to not allow wakeup for specific reasons.
- ECU hardware might support bus speeds 0...1000 kbit/s, but the driver may only work correctly for 500 kbit/s.

The ECU resource template will specify what the HW supports, but the System Template need to take into account the HW plus specific basic SW/configuration that runs on this HW and that may impose further restrictions.
The Connector element is used to describe the bus interfaces of the ECU. It connects the communication port of an ECU with the PhysicalChannel. The Topology is described in more detail in chapter 4.

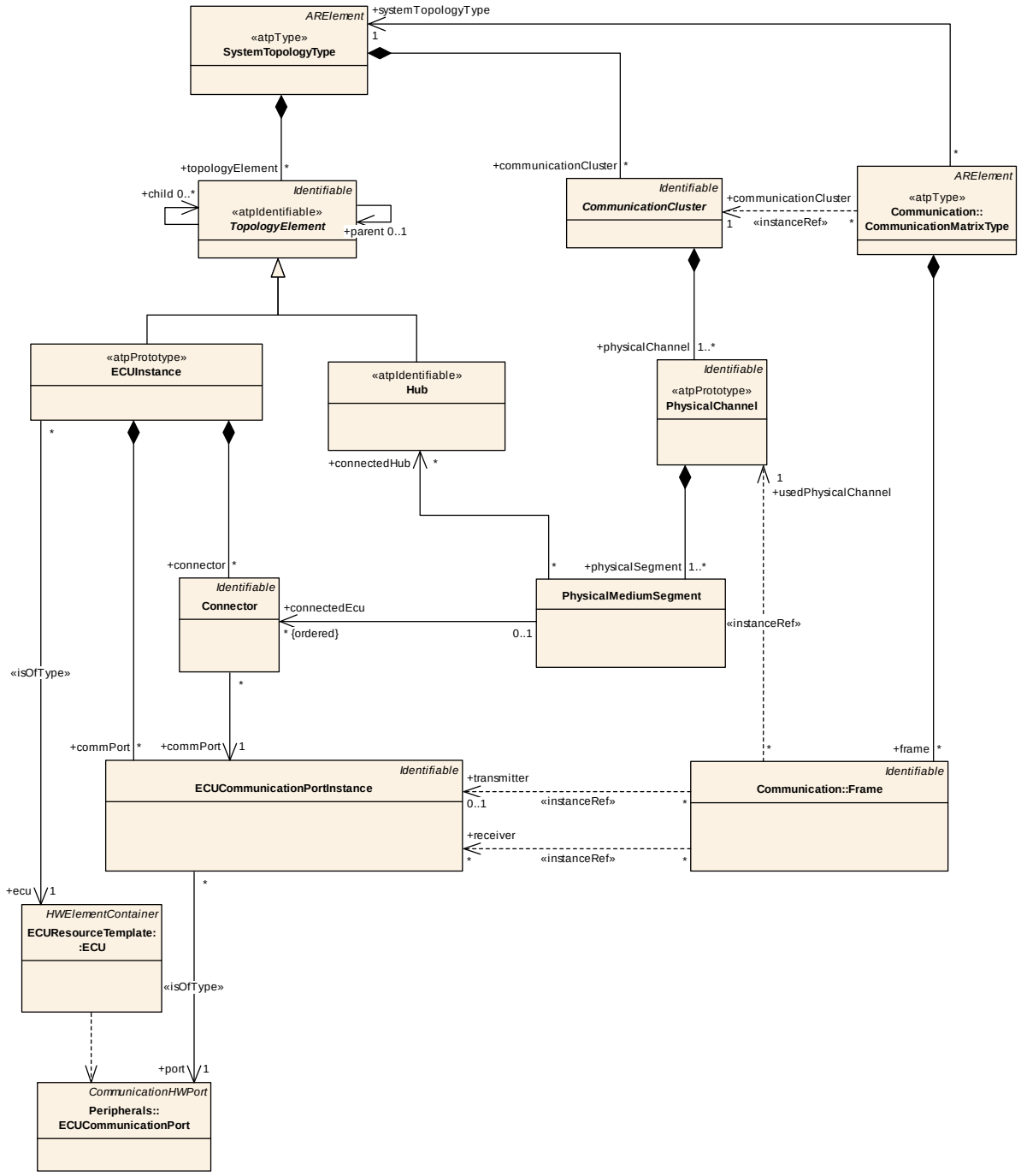

Figure 9: Topology Overview (Topology)

3.4 Communication Overview

The main elements to describe communication in the System Template are the signals, frames, PDUs and communication matrices, as it can be seen on **Figure 10**. The term communication matrix is used to describe all frames exchanged on a communication cluster, and their relation among each other.

The communication matrix is separated from the topology, since the same topology may be used with different communication matrices. The communication matrix is described in more detail in chapter 5. This chapter gives an overview about the frame and signal description.

The SystemSignals allow to represent the communication in a flattened structure, with (at least) one system signal defined for each data element sent or received by a SW component instance. System signals can be defined independently of frames. Furthermore, the SystemSignals are communication cluster independent.

A ClusterSignal represents the SystemSignal on a communication cluster. The SignalPosition element defines the relationship between the ClusterSignal and the SystemSignal.

Furthermore, a ClusterSignal may have aggregated transmitter and receiver specifications. This information can be used for reception filtering.

The PDU (Protocol Data Unit) is a collection of signals for transfer between nodes in a network. The PDUType is merely used to define the PDU structure. The same structure may be used by different PDUPrototypes in different frames. For example, different PDUPrototypes can have the same size and include the same signal layout.

A Frame designates a data frame, which is sent over a physical channel. The aggregation between the Frame and the PDUPrototype allows to define that a frame is composed of several PDUs. For example, it might be possible that one FlexRay Frame contains more than one PDU.

The PDUPrototype contains ClusterSignals which are transmitted within a frame over a communication channel. The receiver ECUs and the transmitter ECUs of a frame are described by the references (transmitter and receiver) to the ECUCommunicationPortInstance class.

The set of receivers defined for the Frame must be a superset of all receivers specified in any ClusterSignal that is included in the frame. All ClusterSignals included in the Frame must also have the same transmitter as the Frame.

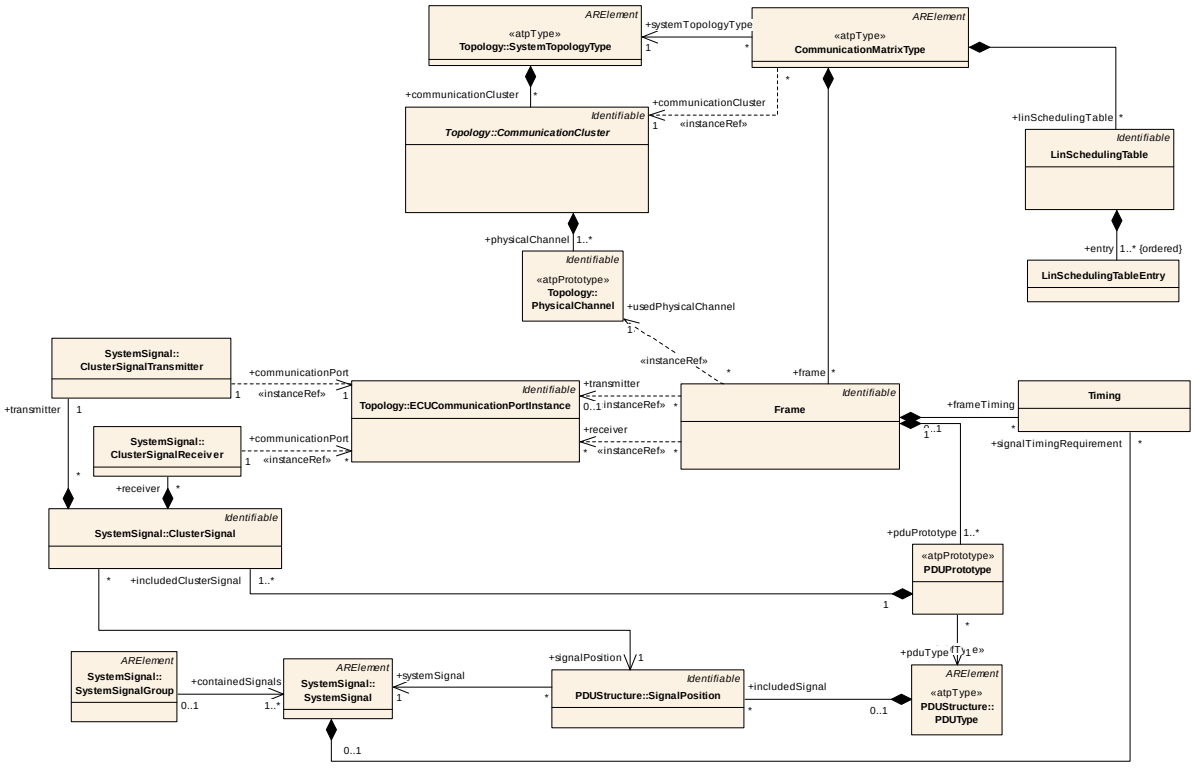

Figure 10: Communication Overview (ComMatrixOverview)

The `Timing` is used for two purposes:

- To define the timing behavior (e.g. Cyclic, Event Triggered) of a frame. One `Frame` can have 0..n Timings. Zero means no timing is specified. If no timing is specified the frame is defined (ID is allocated) but will not be transmitted on the system. (e.g. placeholder Frames). Thus, such a frame has zero transmitters and zero receivers.
- To define the timing requirements (e.g. Cyclic, Event Triggered) of a signal. One `SystemSignal` can have 0..n Timings. Zero means no timing is specified. If no timing is specified no special requirements are defined.

In both cases more than one timing can be defined:

- it is possible that a frame/signal is sent cyclically, and in addition also spontaneously on an occurred event.
- at different ECU states, a different timing can be valid.

The different timing types and the `LinSchedulingTables` will be explained in chapter 5.4.1.2.

3.5 Mapping description and mapping constraints

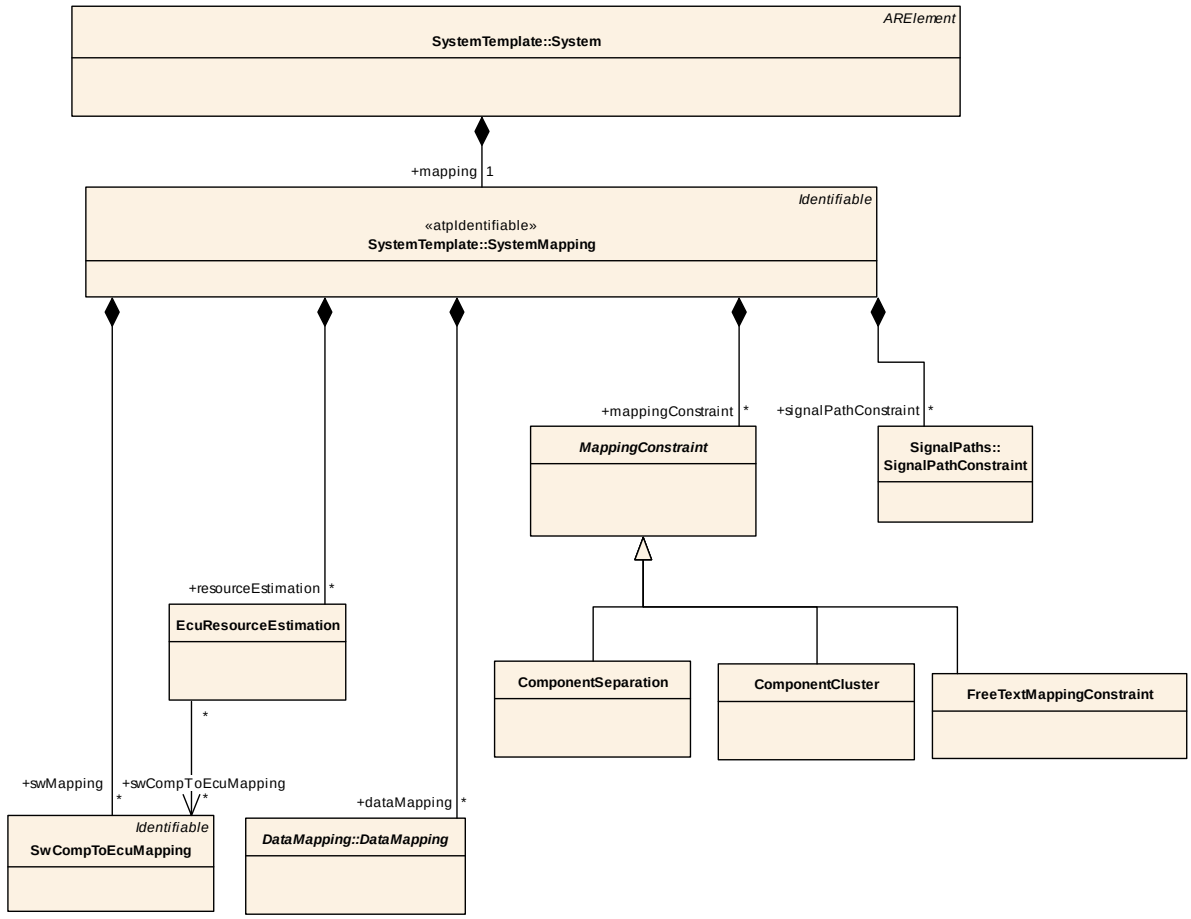

Figure 11: Mapping Description (Mapping)

The System Template allows to describe mappings – both of software components to ECUs as well as data elements exchanged between components to frames, communication protocols or directly to data lines. This is done with the mapping elements, which are aggregated in the `SystemMapping` meta class in the System Template.

The following mappings are defined (compare **Figure 11**):

- The mapping of Software Components to ECUs: `SwCompToEcuMapping` maps one or several software components to ECUs. In the System Constraint Description it is possible to predefine the mapping of software components to ECUs. The predefinition limits the system architect's freedom to map SW components to arbitrary ECUs. After the system generation in the System Configuration Description, all atomic software components that are directly or indirectly part of the top level composition must be mapped with this mapping rule. Software component mapping is described in detail in chapter 6.1.1.
- The `DataMapping` elements are used to map data elements and operations in SW component ports (i.e. the data exchanges between SW components) to signals. The data mapping is described in detail in chapter 6.2.

Furthermore, it is possible to define constraints in the System Constraint Descriptions on the possible SW component mappings, which are aggregated in the `MappingConstraint` elements. `ComponentSeparation` and `ComponentCluster` allow to constraint which SW components must be mapped together on the same ECU (`ComponentCluster`) and which must not be mapped to the same ECU (`ComponentSeparation`).

Mapping constraints that cannot be expressed by any of the other elements can be captured by free text constraints. No tool support will be possible to ensure that natural language constraints are fulfilled, but at least they may be captured in the description.

Details of these mapping constraints are described in chapter 6.1.3. The same chapter contains also additional guidelines for the System Generator, which specific way a signal (data element or client server operation arguments) between two Software Components should take in the network without defining in which frame and with which timing it is transmitted. This Signal Path Constraints are introduced in chapter 6.2.2.

4 Topology

4.1 Scope

The topology describes which ECUs are present in the car, which buses are defined and how AUTOSAR-ECUs and NON-AUTOSAR-ECUs are interconnected. This chapter shows how the System Template of AUTOSAR will describe a car's bus system topology with different bus system properties. The topology is a system constraint for the communication matrix as well as for the mapping of software components.
The topology description is an input for the System Generator and will not be changed by the System Generator. Therefore the Topology is completely described in the System Constraint Description and will be unchanged copied to the System Configuration description. The Topology may only be changed during another iteration development step of the whole system.

4.2 Description of the vehicle's topology

This chapter describes elements and attributes which are needed to describe the vehicle's topology with the System Template.
Figure 12 shows a part of the meta-model which is used to describe a car's bus system topology.
Using the elements `CommunicationCluster`, `PhysicalChannel`, `Hub` and `PhysicalMediumSegment`, different kinds of bus topologies can be described.

Class	**SystemTopologyInstance**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Class Description	The topology of the system. Since several systems may share the same topology, a reference to a system topology type is used to define which topology is used in the system.			
Base Class(es)	ARObject, Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
topology	SystemTopologyType	1	reference to type	

Class	**SystemTopologyInstance**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Class Description	The topology of the system. Since several systems may share the same topology, a reference to a system topology type is used to define which topology is used in the system.			
Base Class(es)	ARObject, Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
topology	SystemTopologyType	1	reference to type	

Class	**SystemTopologyType**			
Package	M2::AUTOSARTemplates::SystemTemplate::Topology			
Class Description	The system topology, which may be reused in different systems.			
Base Class(es)	ARObject, Identifiable, PackageableElement, NonSplitableElement, ARElement			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
communicationCluster	CommunicationCluster	0..*	aggregation	
topologyElement	TopologyElement	0..*	aggregation	

Class	**TopologyElement {abstract}**			
Package	M2::AUTOSARTemplates::SystemTemplate::Topology			
Class Description	A Hub or an ECU. The optional parent-child references can be used for the description of tree topologies. Each element in the tree is appropriately indented to indicate its level in the hierarchy, and is connected to its parent and to the children.			
Base Class(es)	ARObject, Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
child	TopologyElement	0..*	reference	This self-association is optional and is used only for tree topologies. The reference is used to specify the parent-child relationship (The parent node contains a reference to the child node).
parent	TopologyElement	0..1	reference	This self-association is optional and is used only for tree topologies. The reference is used to specify the parent-child relationship (The child node contains a reference to the parent node).

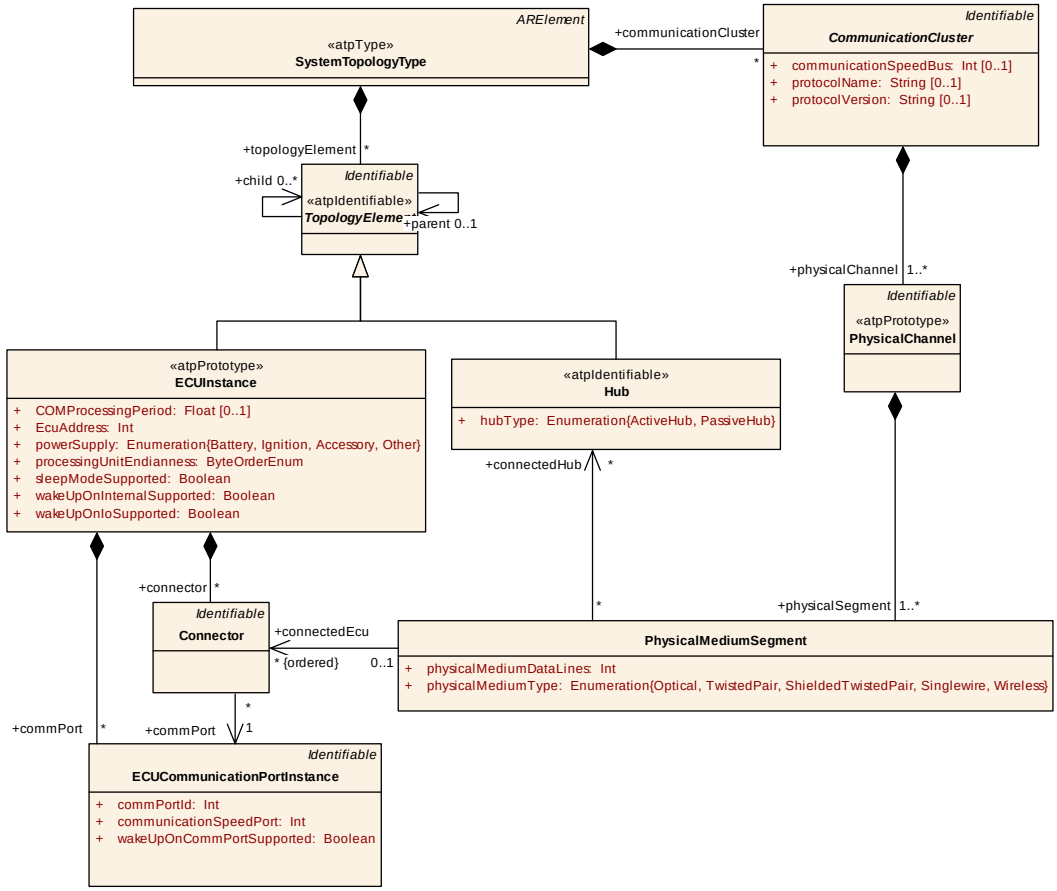

Figure 12: Topology Description (TopologyDetails)

CommunicationCluster: The `CommunicationCluster` is the main element to describe the topological connection of communicating ECUs. An ECU belongs to a communication cluster when a communication port of the ECU is connected to one of the physical channels of the communication cluster. All ECUs within a `CommunicationCluster` communicate within the same address range. A `CommunicationCluster` aggregates one or more `PhysicalChannels`. Note that an ECU can belong to several `CommunicationClusters` if its `CommunicationPorts` are connected to several `PhysicalChannels`.

PhysicalChannel: A `PhysicalChannel` is the transmission medium that is used to send and receive information between two communicating ECUs. Each `CommunicationCluster` has at least one `PhysicalChannel`. Bus systems like CAN and LIN only have exactly one `PhysicalChannel`. A FlexRay cluster may have more than one `PhysicalChannel` that may be used in parallel for redundant communication.

PhysicalMediumSegment: A `PhysicalMediumSegment` defines a set of communicating ECUs, which has the property of providing the identical physical data representation (e.g. electrical potential) to the all connected hardware (ECUs or Hubs). A `PhysicalChannel` may be implemented by several segments, which are then connected via (active or passive) `Hubs`. A `PhysicalMediumSegment` may connect individual ECUs together (for bus topologies), or connects hub and ECU(s) or hubs together. Each `PhysicalChannel` consists of at least one segment; CAN

and LIN have normally only a single segment, FlexRay can consist of more than one segments. These segments are then connected with each other via one or more `Hubs`.

TopologyElement: A Hub or an ECU. In tree topologies the nodes are connected together in a hierarchical manner. The two self-associations on the `TopologyElement` allows to specify parent-child relationships between the nodes, as shown in **Figure 13**. This can be used to describe tree topologies.

Hub: A hub connects several `PhysicalMediumSegments` together. Hubs are only used to describe the detailed bus topology of star and more complex topologies. They are not used on basic bus topologies. A hub can be described as passive or active. A passive hub is just a connection point where the bus system wires are connected together, this information could be e.g. used to analyze whether distributed safety critical application continue work in case of broken wires. In addition the information "active" hub can be used to do a detailed timing analysis.

Connector: A connector is used to describe the bus interfaces of an ECU. Each connector has a reference to exactly one communication port.

ECUCommunicationPortInstance: The ECU Resource Description describes the hardware properties of each `ECUCommunicationPort`. Some of these properties may be further restricted by more software related constraints. Therefore the ECUCommunicationPortInstance describes on one hand the real settings to that hardware and on the other hand some properties, which are set by software only.

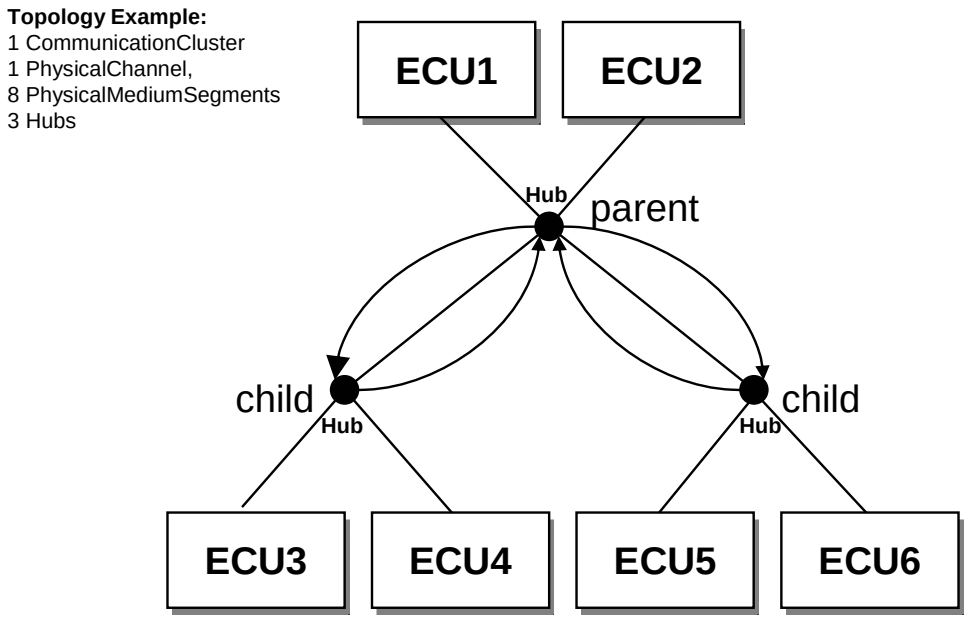

Figure 13: Tree Topology Example

The example in **Figure 14** shows a simple topology with two `CommunicationClusters`. The upper one, the CAN cluster, is modeled as a normal bus. It consists of three `ECUInstances`, each ECU has one CAN `ECUCommunicationPortInstance`. These Ports are connected via one `PhysicalMediumSegment` and one `PhysicalChannel` to a CAN `CommunicationCluster`. The lower one, the FlexRay cluster, is modeled as a redundant star topology. Both physical channels consist of three

PhysicalMediumSegements, which are connected via a Hub to the FlexRay-ECUCommunicationPortInstances.

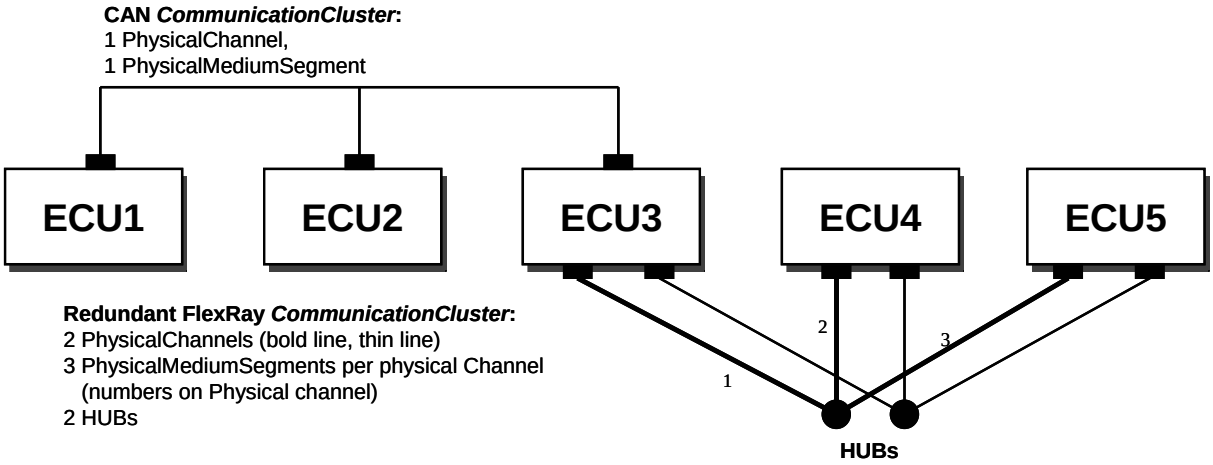

Figure 14: FlexRay Topology Example

In case that two or more SWC which are located on different ECU's need for safety reasons a redundant bus connection, the System Template can provide a detailed topology description. This will be done by one communication cluster, which has more than one PhysicalChannel.
Note that the physical structure of a bus, which consists of several segments and hubs, is in most cases not relevant for the AUTOSAR system generation. The detailed description may be omitted in those cases, and the complete bus may be represented by a single segment. In cases where safety critical software is mapped, or if detailed timing analysis on the network have to be done it might be necessary to have knowledge about the details of the physical architecture, to e.g. make sure that specific software is mapped to ECUs on the same or on different segments to improve availability of the overall system. In those cases, the detailed physical architecture may be described in the System Template.

4.3 General Attributes of the topology entities

The following attributes are valid for all different bus systems. Specific bus system dependant attributes will be described as refinements (chapter 4.4).

4.3.1 Communication Cluster

The following attributes belong to a general communication cluster description. Additional bus system specific attributes are described in the specialized bus system chapters below.

Class	**CommunicationCluster {abstract}**			
Package	M2::AUTOSARTemplates::SystemTemplate::Topology			
Class Description	The CommunicationCluster is the main element to describe the topological connection of communicating ECUs. All ECUs within a CommunicationCluster communicate within the same address range. A CommunicationCluster aggregates one or more physical channels. All physical channels that are aggregated by a communication cluster are synchronized with each other.			
Base Class(es)	ARObject, Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
communicationSpeedBus	Integer	0..1	aggregation	Bus speed in bits per second
physicalChannel	PhysicalChannel	1..*	aggregation	
protocolName	String	0..1	aggregation	The name of the protocol used.
protocolVersion	String	0..1	aggregation	The version of the protocol used.

4.3.2 Physical Channel

These attributes belong to the entity `PhysicalChannel`.

Class	**PhysicalChannel**
Package	M2::AUTOSARTemplates::SystemTemplate::Topology
Class Description	A physical channel is the transmission medium that is used to send and receive information between two communicating ECUs. Each CommunicationCluster has at least one physical channel. Bus systems like CAN and LIN only have exactly one PhysicalChannel. A FlexRay cluster may have more than one PhysicalChannels that may be used in parallel for redundant communication.
Base Class(es)	ARObject, Identifiable

Attribute	*Datatype*	*Mul.*	*Link Type*	*Attribute Description*
physicalSegment	PhysicalMediumSegment	1..*	aggregation	The physical segments that compose the physical channel. CAN and LIN busses that have a simple bus topology will normally have only one segment. All ECUs are connected to this single segment. FlexRay allows for star and other more complex topologies where different ECUs may be connected to different segments and may only communicate via the hub. In those cases, several segments can be defined. Such complex physical structures may be simplified, if the details of the structure are not relevant to the AUTOSAR system generation process. At least a single segment has to be defined, though.

4.3.3 Physical Medium Segment

These attributes belong to the entity `PhysicalMediumSegment`.

Class	**PhysicalMediumSegment**
Package	M2::AUTOSARTemplates::SystemTemplate::Topology
Class Description	A PhysicalMediumSegment defines a set of communicating ECUs, which has the property of providing the identical physical data representation (e.g. electrical potential) to the all connected hardware (ECUs or Hubs). A PhysicalChannel may be implemented by several segments, which are then connected via (active or passive) Hubs. A PhysicalMediumSegment may connect individual ECUs together (for bus topologies), or connects hub and ECU(s) etc. Each PhysicalChannel consists of at least one segment; CAN and LIN have normally only a single segment, FlexRay can consist of more than one segments. These segments are then connected with each other via one or more Hubs.

Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
connectedEcu	Connector	0..*	reference	Reference to the ECU instances connected to the physical medium segment. The ECUs are ordered, since the order of the ECUs on a ring topology is relevant.
connectedHub	Hub	0..*	reference	Reference to the hubs that the segment is connected to. A segment may only connect hubs that are aggregated in the same physical medium.
physicalMediumDataLines	Integer	1	aggregation	Describes the number of physical wires/media used for data transmission. (E.g. single wire CAN has this value set to 1, LIN also 1)
physicalMediumType	PhysicalMediumSegmentPhysical-MediumTypeEnum	1	aggregation	Type of the physical medium on this segment.

4.3.4 Hub

These attributes belong to the entity Hub.

Class	**Hub**
Package	M2::AUTOSARTemplates::SystemTemplate::Topology

Class Description	A hub connects several PhysicalMediumSegments together. Hubs are only used to describe the detailed bus topology of star and more complex topologies. They are not used on basic bus topologies.			
Base Class(es)	ARObject, Identifiable, TopologyElement			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
hubType	HubHubTypeEnum	1	aggregation	The type of hub. Predefined values are "ActiveHub", "PassiveHub"

4.3.5 Connector

These attributes belong to the entity `Connector`.

Class	**Connector**			
Package	M2::AUTOSARTemplates::SystemTemplate::Topology			
Class Description	The connector is used to describe the bus interfaces of the ECU. Each connector has a reference to exactly one communication port. The communication port can be referenced by several connector elements. This is important for the FlexRay Bus. FlexRay communicates via two physical channels. But only one controller in an ECU is responsible for both channels. Thus, two connectors (for channel A and for channel B) must reference to the same communication port.			
Base Class(es)	ARObject, Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
commPort	ECUCommunicationPortInstance	1	reference	Reference to the communication port.

4.4 Specialized attributes of the topology entities

Some communication clusters need beside the general attributes, which are valid for all communication clusters, additional attributes to describe the individual communication cluster properties.

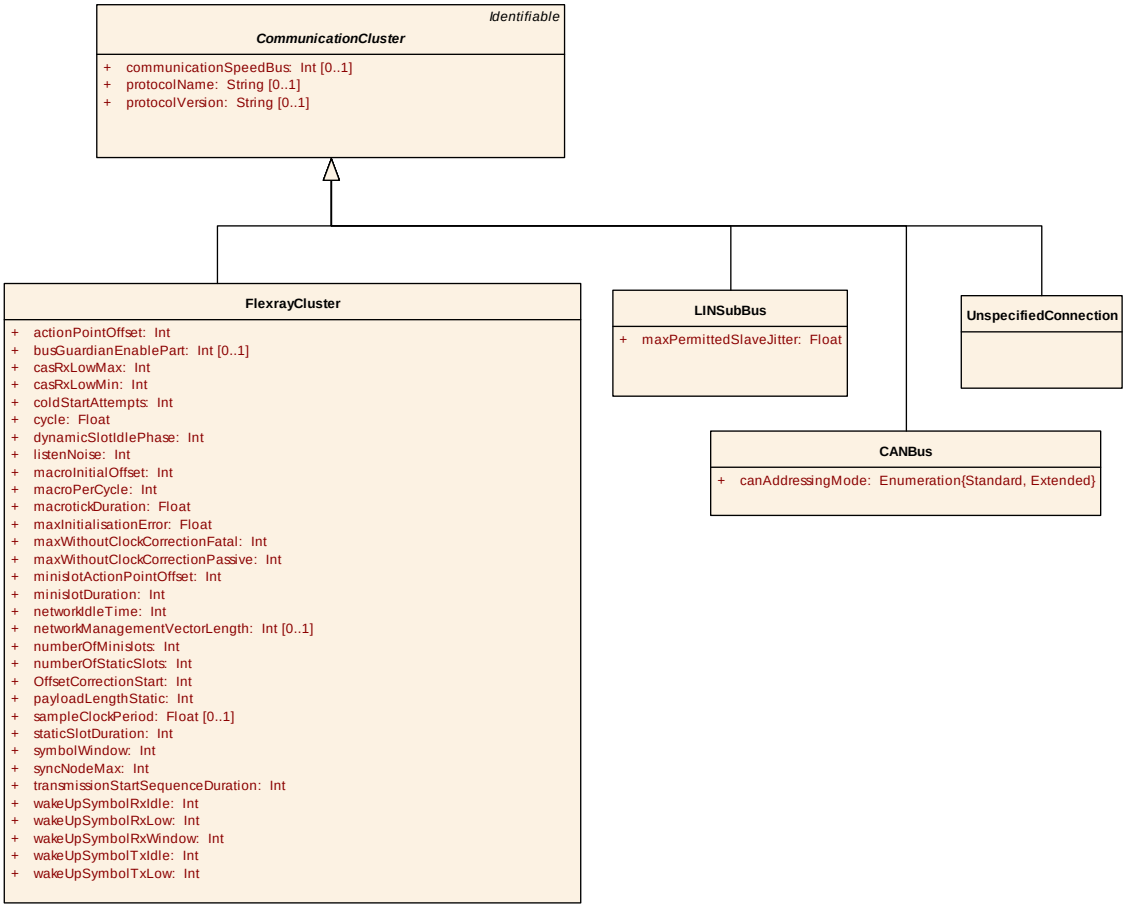

Figure 15: Specialized Topology Attributes (TopologyAttributeRefinement)

4.4.1 CAN Bus

CAN specific attributes of the element `CommunicationCluster`.

Class	**CANBus**
Package	M2::AUTOSARTemplates::SystemTemplate::Topology
Class Description	CAN specific attributes
Base Class(es)	ARObject, Identifiable, CommunicationCluster

Attribute	*Datatype*	*Mul.*	*Link Type*	*Attribute Description*
canAddressingMode	CANBusCanAddressingModeEnum	1	aggregation	The CAN bus supports 11-Bit ("Standard") and 29-Bit ("Extended") identifiers. This attributes constrains a CAN bus to the selected formats. On Extended-Addressing it is also possible to have 11-Bit and 29-Bit CAN-identifiers. Predefined values are "Standard" and "Extended".

4.4.2 FlexRay Cluster

FlexRay specific attributes of element `CommunicationCluster`. The attributes are based on the FlexRay specification 2.0.

Class	**FlexrayCluster**			
Package	M2::AUTOSARTemplates::SystemTemplate::Topology			
Class Description	FlexRay specific attributes to the physical Medium			
Base Class(es)	ARObject, Identifiable, CommunicationCluster			
Attribute	*Datatype*	*Mul.*	*Link Type*	*Attribute Description*
actionPointOffset	Integer	1	aggregation	The offset of the action point in networks
busGuardianEnablePart	Integer	0..1	aggregation	Bus Guardian Inter Slot Gap (ISG) part that follows a guarded schedule element. Unit macroticks
casRxLowMax	Integer	1	aggregation	Upper limit of the Collision Avoidance Symbol (CAS) acceptance window. Unit:bitDuration

casRxLowMin	Integer	1	aggregation	Lower limit of the Collision Avoidance Symbol (CAS) acceptance window. Unit:bitDuration
coldStartAttempts	Integer	1	aggregation	The maximum number of times that a node in this cluster is permitted to attempt to start the cluster by initiating schedule synchronization
cycle	Float	1	aggregation	Length of the cycle. Unit: seconds
dynamicSlotIdlePhase	Integer	1	aggregation	The duration of the dynamic slot idle phase in minislots.
listenNoise	Integer	1	aggregation	Upper limit for the start up and wake up listen timeout in the presence of noise. Expressed as a multiple of the cluster constant pdListenTimeout. Unit microticks
macroInitialOffset	Integer	1	aggregation	number of macroticks which describe the distance between the static slot boundary and the closed macrotick boundary of the secondary time reference point using the initial configured macrotick length
macroPerCycle	Integer	1	aggregation	The number of macroticks in a communication cycle
macrotickDuration	Float	1	aggregation	The duration of the cluster wide nominal macrotick. Unit: seconds
maxInitialisationError	Float	1	aggregation	The maximum error that a node may have after initialization. Unit: seconds
maxWithoutClockCorrectionFatal	Integer	1	aggregation	Threshold concerning vClockCorrectionFailedCounter. Defines when the Communication Controller (CC) shall change from normal or passive state to hold state.
maxWithoutClockCorrection-Passive	Integer	1	aggregation	Threshold concerning vClockCorrectionFailedCounter. Defines when the Communication Controller (CC) shall change from normal or passive state to hold state.
minislotActionPointOffset	Integer	1	aggregation	The Offset of the action point within a minislot. Unit: macroticks
minislotDuration	Integer	1	aggregation	The duration of a minislot (dynamic segment). Unit: macroticks.
networkIdleTime	Integer	1	aggregation	The duration of the network idle time in macroticks
networkManagementVectorLength	Integer	0..1	aggregation	Length of the Network Management vector on a cluster. Unit: Bytes
numberOfMinislots	Integer	1	aggregation	number of Minislots in the dynamic segment.
numberOfStaticSlots	Integer	1	aggregation	The number of static slots in the static segment.
OffsetCorrectionStart	Integer	1	aggregation	Start of the offset correction phase within the Network Idle Time (NIT), expressed as the number of macroticks from the

				start of cycle. Unit: macroticks
payloadLengthStatic	Integer	1	aggregation	Globally configured payload length of a static frame. Unit: 16-bit WORDS.
sampleClockPeriod	Float	0..1	aggregation	Sample clock period. Unit: seconds
staticSlotDuration	Integer	1	aggregation	The duration of a slot in the static segment. Unit: macroticks
symbolWindow	Integer	1	aggregation	The duration of the symbol window. Unit: macroticks
syncNodeMax	Integer	1	aggregation	The maximum number of sync nodes allowed in the cluster
transmissionStartSequence-Duration	Integer	1	aggregation	Duration of the Transmission Start Sequence. Unit: bit
wakeUpSymbolRxIdle	Integer	1	aggregation	Number of bits used by the node to test the duration of the idle portion of a received wake up symbol. Unit:bitDuration
wakeUpSymbolRxLow	Integer	1	aggregation	Number of bits used by the node to test the LOW portion of a received wake up symbol. Unit:bitDuration
wakeUpSymbolRxWindow	Integer	1	aggregation	Number of bits used by a node to test the overall duration of a received wake up symbol. Unit: gdBit
wakeUpSymbolTxIdle	Integer	1	aggregation	Number of bits used by the node to transmit the idle part of a wake up symbol. Unit: gDbit
wakeUpSymbolTxLow	Integer	1	aggregation	Number of bits used by the node to transmit the LOW part of a wake up symbol. Unit:bitDuration

4.4.3 LIN SubBus

The `LINSubBus` does not have any specific attributes, but it simplifies future extensions.

4.4.4 Unspecified Connection

The `UnspecifiedConnection` may be used for proprietary bus systems, direct connections between I/O-ports of ECUs etc. The `UnspecifiedConnection` does not have any specific attributes, but it simplifies future extensions.

4.5 Communication Port Instances

This chapter describes entities and attributes which are needed to describe the `ECUCommunicationPortInstance` of an ECU.

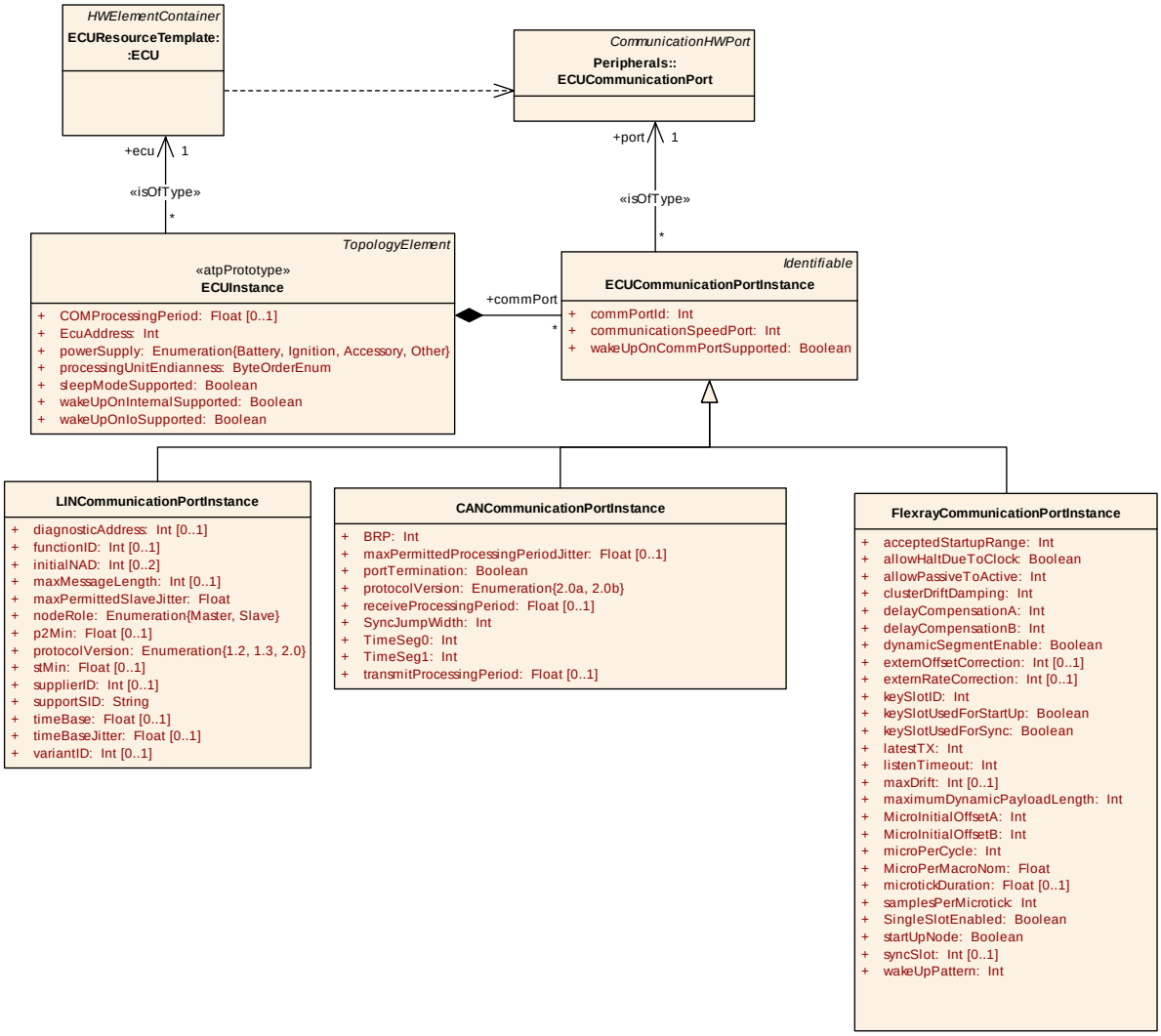

Figure 16: ECUCommunicationPortInstance Details (ECUInstanceDetails)

4.5.1 ECUCommunicationPortInstance

These attributes are valid for all different kinds of communication ports. Bus system dependent attributes are described as refinements at the chapters below.

Class	**ECUCommunicationPortInstance**			
Package	M2::AUTOSARTemplates::SystemTemplate::Topology			
Class Description	The ECU resource description describes the hardware properties of each ECU communication port. Some of these properties may be further restricted by more software related constraints. Therefore the communication port instance describes on one hand the real settings to that hardware and on the other hand some properties which are set by software only.			
Base Class(es)	ARObject, Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
commPortId	Integer	1	aggregation	The identifier for this communication port on the bus, which can be used as e.g. network management address. There must not be two identical communication port identifiers on the same communication cluster.
communicationSpeedPort	Integer	1	aggregation	Bus speed of this port in bits per second

port	ECUCommunicationPort	1	reference to type	Reference to the port in the ECU description that is used. The reference from ECUInstance to ECU and from the aggregated ECUCommunicationPortInstance to ECUCommunicationPort must match and point to the same ECU: If ECUInstance X refers to ECU Y, then ECUCommunicationPortInstance X/P must refer to an ECUCommunicationPort of Y, e.g. Y/Q.
wakeUpOnCommPortSupported	Boolean	1	aggregation	May the ECU be woken up on this communication port? TRUE: wake up is possible FALSE: wake up on bus is not supported Note: This flag may only be set to TRUE if the feature is supported by both hardware and basic software.

4.5.2 LINCommunicationPortInstance

Communication port attributes which are only used for LIN communication ports.

Class	**LINCommunicationPortInstance**			
Package	M2::AUTOSARTemplates::SystemTemplate::Topology			
Class Description	LIN bus specific communication port instance attributes.			
Base Class(es)	ARObject, Identifiable, ECUCommunicationPortInstance			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
diagnosticAddress	Integer	0..1	aggregation	Diagnostic address of the LIN node. This attribute is only use for LIN 1.2 and LIN 1.3 slaves.

functionID	Integer	0..1	aggregation	function_id is a 16 bit number assigned to the product by the supplier to make it unique.
initialNAD	Integer	0..2	aggregation	Unique ID used e.g. for automatic configuration of LIN networks. The Node Address for Diagnostic (NAD) is not part of the LIN 1.2 and 1.3 specification. The NAD is only assigned to LIN slaves.
maxMessageLength	Integer	0..1	aggregation	The max_message_length property applies to the diagnostic transport layer only; it defines the maximum length of a diagnostic message in bits.
maxPermittedSlaveJitter	Float	1	aggregation	Maximum jitter time of responding frames, only needed for LIN slaves. Unit is seconds.
nodeRole	LINCommunicationPortInstance-NodeRoleEnum	1	aggregation	Specifies whether the LIN node is master or slave on the bus it is connected to with this communication port
p2Min	Float	0..1	aggregation	P2_min specifies the minimum time in seconds between a master request frame and the following slave response frame for the node to be able to prepare the response.
protocolVersion	LINCommunicationPortInstance-ProtocolVersionEnum	1	aggregation	The version of the protocol used. Predefined values for LIN are "1.2", "1.3", "2.0".
stMin	Float	0..1	aggregation	ST_min specifies the minimum time in seconds between two slave response frames in a multi-PDU response, i.e. it only applies to the diagnostic transport layer.
supplierID	Integer	0..1	aggregation	Unique supplier ID of the LIN slave. A unique ID is assigned to every member of the LIN consortium. It may be used for the automatic

				configuration of LIN 2.0 slave nodes.
supportSID	String	1	aggregation	Lists all Service Identifier (SID) values that are supported by the node. Example: {0xb0, 0xb1, 0xb2}
timeBase	Float	0..1	aggregation	Time base is mandatory for the master. It is not used for slaves. LIN 2.0 Spec states: "The time_base value specifies the used time base in the master node to generate the maximum allowed frame transfer time." The time base shall be specified AUTOSAR conform in seconds.
timeBaseJitter	Float	0..1	aggregation	timeBaseJitter is a mandatory attribute for the master and not used for slaves. LIN 2.0 Spec states: "The jitter value specifies the differences between the maximum and minimum delay from time base start point to the frame header sending start point (falling edge of BREAK signal)." The jitter shall be specified AUTOSAR conform in seconds.
variantID	Integer	0..1	aggregation	The Variant ID is used for the product identification and is an 8 bit value specifying the variant. The variant ID shall be changed whenever the product is changed but with unaltered function.

4.5.3 CANCommunicationPortInstance

Communication port attributes which are only used for `CANCommunicationPortInstances`.

Class	**CANCommunicationPortInstance**			
Package	M2::AUTOSARTemplates::SystemTemplate::Topology			
Class Description	CAN bus specific communication port attributes.			
Base Class(es)	ARObject, Identifiable, ECUCommunicationPortInstance			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
BRP	Integer	1	aggregation	A clock prescaler value
maxPermittedProcessingPeriod-Jitter	Float	0..1	aggregation	Specifies the maximum acceptable variance (normally delay) in seconds from the nominal Processing Period of the call to execute the Bus Communication SW. The cause of the jitter may be interrupts from tasks in the CPU with higher priority. This requirement must be taken into account when scheduling the software in the CPU.
portTermination	Boolean	1	aggregation	Describes whether the transceiver communication port termination is on or off. TRUE: communication port is terminated FALSE: communication port is not terminated

protocolVersion	CANCommunicationPortInstance-ProtocolVersionEnum	1	aggregation	Describes whether the CAN communication port is CAN "2.0a" or "2.0b" compliant.
receiveProcessingPeriod	Float	0..1	aggregation	Specifies the nominal time interval between two consecutive calls to execute the Bus Communication SW (the COM layer) for receiving. Unit: seconds. At the execution of the Bus Communication SW received frames are moved from receive buffers and their signal content made available to the application.
SyncJumpWidth	Integer	1	aggregation	The number of quanta in the Synchronization Jump Width, SJW
TimeSeg0	Integer	1	aggregation	The number of quanta before the sampling point
TimeSeg1	Integer	1	aggregation	The number of quanta after the sampling point
transmitProcessingPeriod	Float	0..1	aggregation	Specifies the nominal time interval between two consecutive calls to execute the Bus Communication SW (the COM layer) for sending. Unit: seconds. At the execution of the Bus Communication SW frames due for transmission are moved to the transmit buffers, and made available for arbitration on the bus (in the case of CAN).

4.5.4 FlexRayCommunicationPortInstance

Communication port attributes which are only used for `FlexrayCommunicationPortInstances`.

Class	**FlexrayCommunicationPortInstance**			
Package	M2::AUTOSARTemplates::SystemTemplate::Topology			
Class Description	FlexRay bus specific communication port attributes.			
Base Class(es)	ARObject, Identifiable, ECUCommunicationPortInstance			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
acceptedStartupRange	Integer	1	aggregation	Expanded range of measured clock deviation allowed for startup frames during integration. Unit:microtick
allowHaltDueToClock	Boolean	1	aggregation	Boolean flag that controls the transition to the POC:halt state due to a clock synchronization errors. If set to true, the Communication Controller is allowed to transition to POC:halt. If set to false, the Communication Controller will not transition to the POC:halt state but will enter or remain in the normal POC (passive State).
allowPassiveToActive	Integer	1	aggregation	Number of consecutive even/odd cycle pairs that must have valid clock correction terms before the Communication Controller will be allowed to transition from the POC:normal passive state to POC:normal active state. If set to 0, the Communication Controller is not allowed to transition from POC:norm
clusterDriftDamping	Integer	1	aggregation	The cluster drift damping factor used in clock synchronization rate correction in microticks
delayCompensationA	Integer	1	aggregation	Value used to compensate for reception delays on channel A
delayCompensationB	Integer	1	aggregation	Value used to compensate for reception delays on channel B
dynamicSegmentEnable	Boolean	1	aggregation	Boolean flag that configures the Bus Guardian Schedule Monitoring Service to expect transmissions within the dynamic segment.

externOffsetCorrection	Integer	0..1	aggregation	Fixed amount added or subtracted to the calculated offset correction term to facilitate external offset correction, expressed in node-local microticks.
externRateCorrection	Integer	0..1	aggregation	Fixed amount added or subtracted to the calculated rate correction term to facilitate external rate correction, expressed in node-local microticks.
keySlotID	Integer	1	aggregation	ID of the slot used to transmit the startup frame, sync frame, or designated single slot frame.
keySlotUsedForStartUp	Boolean	1	aggregation	Flag indicating whether the Key Slot is used to transmit a startup frame.
keySlotUsedForSync	Boolean	1	aggregation	Flag indicating whether the Key Slot is used to transmit a sync frame.
latestTX	Integer	1	aggregation	The number of the last minislot in which a transmission can start in the dynamic segment for the respective node
listenTimeout	Integer	1	aggregation	Upper limit for the start up listen timeout and wake up listen timeout.
maxDrift	Integer	0..1	aggregation	Maximum drift offset in microticks between two nodes that operate with unsynchronized clocks over one communication cycle.
maximumDynamicPayloadLength	Integer	1	aggregation	Maximum payload length for the dynamic channel of a frame in 16 bit WORDS.
MicroInitialOffsetA	Integer	1	aggregation	Number of microticks between the closest macrotick boundary described by gMacroInitialOffset and the secondary time reference point. The parameter depends on pDelayCompensationA and therefore it has to be set independently for each channel.
MicroInitialOffsetB	Integer	1	aggregation	Number of microticks between the closest macrotick boundary described by gMacroInitialOffset and the secondary time reference point. The parameter depends on pDelayCompensationB and therefore it has to be set independently for each channel.
microPerCycle	Integer	1	aggregation	The nominal number of microticks in a communication cycle
MicroPerMacroNom	Float	1	aggregation	Number of microticks per nominal macrotick that all implementations must support.
microtickDuration	Float	0..1	aggregation	Duration of a microtick. This attribute can be derived from samplePerMicrotick and gdSampleClockPeriod. Unit: seconds
samplesPerMicrotick	Integer	1	aggregation	Number of samples per microtick

SingleSlotEnabled	Boolean	1	aggregation	Flag indicating whether or not the node shall enter single slot mode following startup.
startUpNode	Boolean	1	aggregation	Indicates that the node is a startup node (startup frame configured; connected to gChannels)
syncSlot	Integer	0..1	aggregation	The number of the static slot in which a sync frame shall be sent, if a sync frame shall be sent
wakeUpPattern	Integer	1	aggregation	Number of repetitions of the Tx-wakeup symbol to be sent during the CC_WakeupSend state of this Node in the cluster

4.6 Direct Data Lines

The topology description is not limited to the description of bus systems like CAN, LIN and so on. It is also possible to describe direct data lines between ECUs.
For those data lines the communication cluster `UnspecifiedConnection` has to be used. Each direct data line consists of one `CommunicationCluster`, one `PhysicalChannel`, one `PhysicalMediumSegment` and the connections to the ECUs.

Class	**UnspecifiedConnection**
Package	M2::AUTOSARTemplates::SystemTemplate::Topology
Class Description	UnspecifiedConnection may be used for proprietary bus systems, direct connections between I/O-ports of ECUs etc.
Base Class(es)	ARObject, Identifiable, CommunicationCluster

4.7 Description of ECUInstance attributes

At the `ECUInstance`, the software configurable properties of the ECU are described. The following table describes the attributes, which are needed to describe bus system, power supply and activation ports.

Class	**ECUInstance**			
Package	M2::AUTOSARTemplates::SystemTemplate::Topology			
Class Description	ECUInstances are used to define the ECUs used in the topology. The type of the ECU is defined by a reference to an ECU specified with the ECU resource description.			
Base Class(es)	ARObject, Identifiable, TopologyElement			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
COMProcessingPeriod	Float	0..1	aggregation	The COM scheduling time is used in order to be able to calculate the worst case bus timing. The processing period shall be specified AUTOSAR conform in seconds.
commPort	ECUCommunication-PortInstance	0..*	aggregation	
connector	Connector	0..*	aggregation	
EcuAddress	Integer	1	aggregation	The unique address of the ECU within the car. Used e.g. for Onboard Diagnostic (OBD) addressing, general diagnostics, Network Management.
ecu	ECU	1	reference to type	Reference to the ECU hardware used for this ECUInstance. The reference from ECUInstance to ECU and from the aggregated ECUCommunicationPortInstance to ECUCommunicationPort must match and point to the same ECU: If ECUInstance X refers to ECU Y, then ECUCommunicationPortInstance X/P must refer to an ECUCommunicationPort of Y, e.g. Y/Q.

powerSupply	ECUInstance-PowerSupplyEnum	1	aggregation	Defines how the ECU is connected to power feed. Predefined values are Battery: the ECU is always connected to power Ignition: the ECU has power feed if ignition is on Accessory: the ECU has power feed if accessory is switched on. Other: other power feed mode, such as selective power feed, e.g. under control of another ECU, etc.
processingUnitEndianness	ByteOrderEnum	1	aggregation	The Processing Unit endianness is the byte order in which the CPU interprets multi-byte integers from memory. The byte ordering "Little Endian" (MostSignificantByteLast) and "Big Endian" (MostSignificantByteFirst) can be selected.
sleepModeSupported	Boolean	1	aggregation	Specifies whether the ECU instance may be put to a "low power mode" TRUE: sleep mode is supported FALSE: sleep mode is not supported Note: This flag may only be set to TRUE if the feature is supported by both hardware and basic software.
wakeUpOnInternalSupported	Boolean	1	aggregation	Specifies whether the ECU may be woken from sleep mode on internal events, e.g. by timer or similar device built into the ECU. TRUE: ECU may wake up on ECU internal event FALSE: ECU does not support this feature Note: This flag may only be set to TRUE if the feature is supported by both hardware and basic software.
wakeUpOnIoSupported	Boolean	1	aggregation	Specifies whether the ECU may be woken up on IO ports (other than communication ports - those are specified separately). TRUE: ECU may be woken up on IO ports (see ECU resource descriptions for details) FALSE: not supported Note: This flag may only be set to TRUE if the feature is supported by both hardware and basic software.

5 Communication

This chapter describes all topics that deal with constraints or configurations that describe the information exchange between the ECUs. The two main elements to describe communication in the System Template are the signals and communication matrices.
A communication matrix describes communication activities between ECUs of one communication cluster. The communication matrix is composed of frames, which are in turn composed of PDUs. The PDUs are composed of individual signals. The communication matrix also describes the parameters (e.g. the timing) of the frames and its transmitters and receivers.

5.1 Signals

`SystemSignals` can be defined independently of the frames and the communication clusters. On the communication cluster they are represented by the referencing `ClusterSignal`. `ClusterSignals` describe the cluster specific requirements of the sender and receiver communication.
In the metamodel the element `SignalPosition` creates the relationship between a `SystemSignal` and a `ClusterSignal`. The `SignalPosition` within a `PDUType` refers to the `SystemSignal` that shall be transported at this position in the PDU. The `ClusterSignal` is part of a `PDUPrototype` and refers to the `SignalPosition`.
The `SystemSignals`, together with the `ClusterSignals`, represent the signals used in AUTOSAR COM. The name used in the definition of the `SystemSignal` will be displayed in network monitoring systems which trace network activity.
The referencing from a `ClusterSignal` to the `EcuCommunicationPortInstances` that transmit and receive the signal is not done with a direct reference, but with two “helper classes” `ClusterSignalTransmitter` and `ClusterSignalReceiver`. This allows to define attributes specific for a single transmitter or receiver of the signal. For example, attribute timeout may be set individually for each receiver of a `ClusterSignal`.
Consequently, the attributes defined in `ClusterSignalTransmitter` and `ClusterSignalReceiver` define the behavior of the referenced ECU communication port.

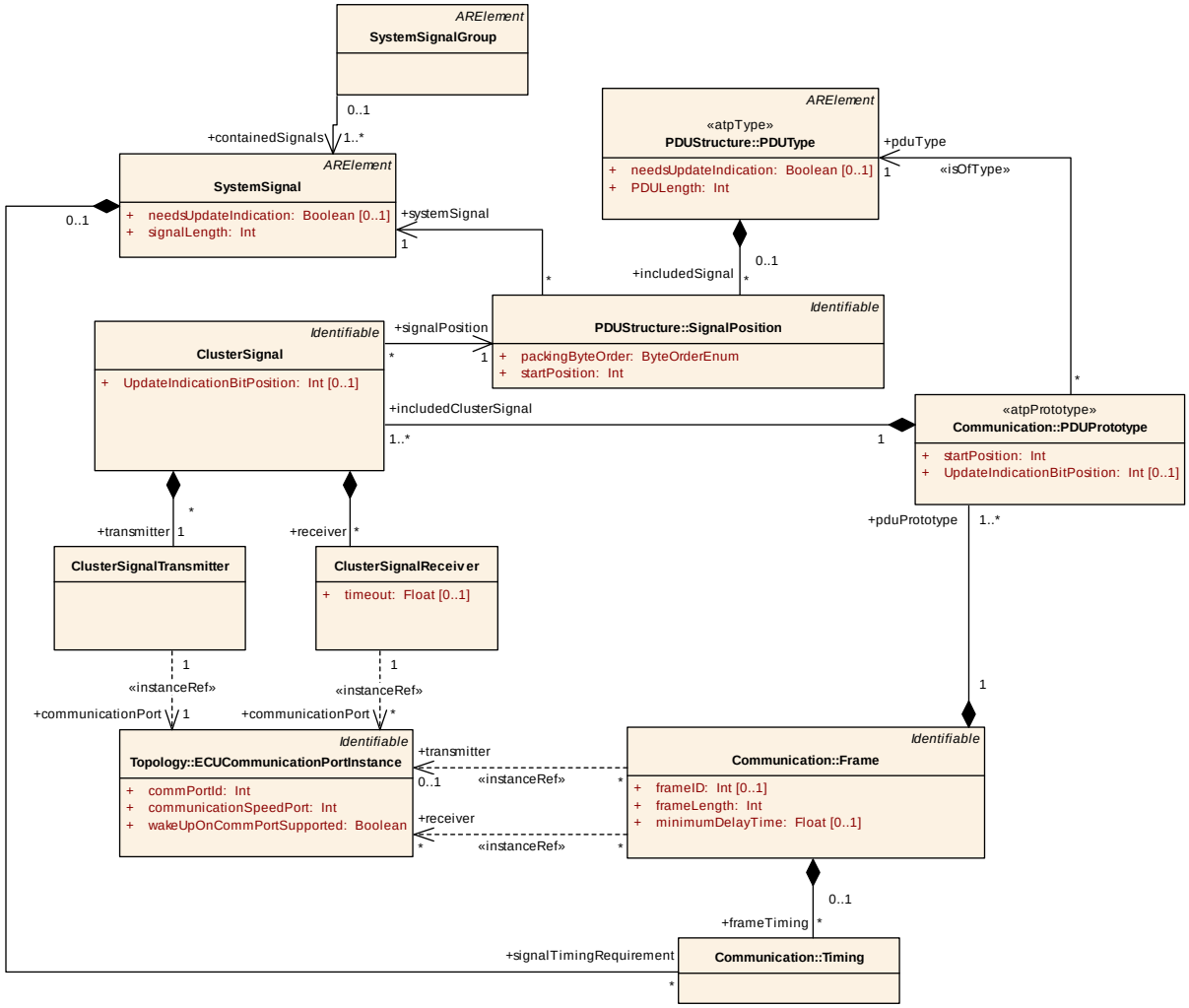

Figure 17: System Signal and Cluster Signal (SystemSignal)

Class	**SystemSignal**			
Package	M2::AUTOSARTemplates::SystemTemplate::SystemSignal			
Class Description	The system signal represents the communication system's view of data exchanged between SW components which reside on different ECUs. The system signals allow to represent this communication in a flattened structure, with (at least) one system signal defined for each data element sent or received by a SW component instance. If data has to be sent over gateways, there is still only one system signal representing this data. The representation of the data on the individual communication systems is done by the cluster signals. System signals can be defined independently of frames and PDUs. This allows a development methodology where the signals are defined first, and are assigned to frames in a later stage. According to the COM Specification, signal groups without signals are allowed. These have a "signalLength" = 0. In this case there shall be an "update-bit" configured.			
Base Class(es)	ARObject, Identifiable, PackageableElement, NonSplitableElement, ARElement			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
needsUpdateIndication	Boolean	0..1	aggregation	Boolean value that defines whether a UpdateIndicationBit has to be defined in the PDU that is associated with this SystemSignal. If this attribute is omitted, no UpdateIndication bit will be added to the PDU.
signalLength	Integer	1	aggregation	The signalLength specifies the size of the signal in bits.
signalTimingRequirement	Timing	0..*	aggregation	A SystemSignal can have aggregated timing requirements. These timing requirements have to be taken into account when several cluster signals are grouped together into a frame.

Class	**ClusterSignal**			
Package	M2::AUTOSARTemplates::SystemTemplate::SystemSignal			
Class Description	A cluster signal represents the system signal on one specific communication cluster.			
Base Class(es)	ARObject, Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
receiver	ClusterSignalReceiver	0..*	aggregation	The receivers of the cluster signal. All transmitters and receivers of a cluster signal must be located on the same communication cluster.
signalPosition	SignalPosition	1	reference	Reference to the signal position. The system signal at this position is represented on the communication cluster by the referencing cluster signal.
transmitter	ClusterSignalTransmitter	1	aggregation	The transmitters of the cluster signal. All transmitters and receivers of a cluster signal must be located on the same communication cluster.
UpdateIndicationBitPosition	Integer	0..1	aggregation	The UpdateIndicationBit indicates to the receivers that the clusterSignal was updated by the sender. Length is always one bit. The UpdateIndicationBitPosition attribute describes the position of the update bit within the PDU.

Class	**ClusterSignalTransmitter**			
Package	M2::AUTOSARTemplates::SystemTemplate::SystemSignal			
Class Description	A transmitter of the cluster signal. Since different transmitters can have different communication attribute values, a separate class is used per transmitter.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
communicationPort	ClusterSignalTransmitter_communication-Port	1	reference to instance	Reference to the transmitting communication port.

Class	**ClusterSignalReceiver**			
Package	M2::AUTOSARTemplates::SystemTemplate::SystemSignal			
Class Description	A receiver of the cluster signal. Since different receivers can have different communication attribute values, a separate class is used per receiver.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
communicationPort	ClusterSignalReceiver_communication-Port	0..*	reference to instance	
timeout	Float	0..1	aggregation	Timeout in seconds. If the timeout occurs without receiving the signal, the ECU shall take appropriate actions.

5.2 PDU Type

The PDU defines a collection of signals for transfer between nodes in a network. The PDU is described by two elements. With the `PDUType` element it is possible to define the structure of a PDU. Therewith, several different `PDUPrototypes` can use the same structure. This is described by the <<isofType>> reference between the `PDUType` and the `PDUPrototype` class. The `PDUPrototypes` are part of a `Frame` and are described in more detail in chapter 5.4.1.3.
The signals, which are contained within the PDU must have a `SignalPosition`. The `SignalPosition` element contains a reference to the `SystemSignal` element, which is transported at this position in the PDU.
A PDU can also contain multiplexed data. This is realized by the `Multiplexer` class.

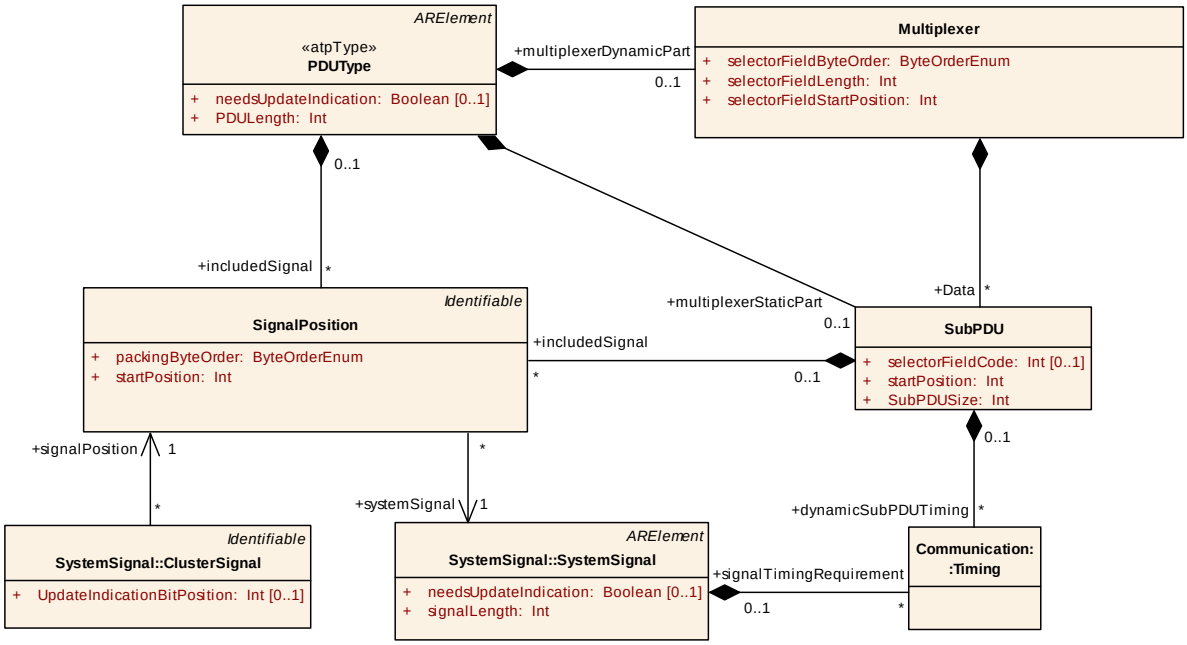

Figure 18: PDUType and the SignalPosition (PDUStructure)

Class	**PDUType**			
Package	M2::AUTOSARTemplates::SystemTemplate::PDUStructure			
Class Description	PDUType defines the structure of a PDU.			
Base Class(es)	ARObject, Identifiable, PackageableElement, NonSplitableElement, ARElement			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
includedSignal	SignalPosition	0..*	aggregation	Definition of the signals included in the PDU with their position. All signals included must have the same transmitter (as specified in the cluster signal).
multiplexerDynamicPart	Multiplexer	0..1	aggregation	Multiplexing is used to transport varying signals at the same position of a single PDU. According to the value of the selector field some parts of the PDU have a different layout. These parts of the PDU that can contain different signals are called dynamic part.
multiplexerStaticPart	SubPDU	0..1	aggregation	Some parts of the PDU may be the same regardless of the selector field. Such a part is called static part. It is possible that the static part of a PDU is zero bits long (only the dynamic part is defined).
needsUpdateIndication	Boolean	0..1	aggregation	Boolean value that defines whether a UpdateIndicationBit has to be defined in the Frame that is associated with this PDUType. If this attribute is omitted, no UpdateIndication bit will be added to the Frame. The AUTOSAR FlexRay Interface explicitly defines a location for the update bit that relates to a PDU.
PDULength	Integer	1	aggregation	The size of the PDU in bits. The size is limited by the frameLength.

5.2.1 Signal Position

The `SignalPosition` describes a position of a signal within a PDU. The `SignalPosition` definition is defined as follows:

Class	**SignalPosition**			
Package	M2::AUTOSARTemplates::SystemTemplate::PDUStructure			
Class Description	The signal position defines which system signal is transmitted at which position within the aggregating PDU. The SignalPosition must be defined within a PDU or a SubPDU.			
Base Class(es)	ARObject, Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
packingByteOrder	ByteOrderEnum	1	aggregation	This parameter defines the order of the bytes of the signal are packed into the PDU. The byte ordering "Little Endian" (MostSignificantByteLast) and "Big Endian" (MostSignificantByteFirst) can be selected. The value of this attribute impacts the absolute position of the signal into the PDU (see the startPosition attribute description).
startPosition	Integer	1	aggregation	This parameter is necessary to describe the position of a signal within a PDU or SubPDU. It denotes the least significant bit for "Little Endian" and the most significant bit for "Big Endian" packed signals within the PDU (see the description of the packingByteOrder attribute). Bits within the PDU are counted as follows (see the OSEK COM v3.0.3 specification) : Bit 0 corresponds to Byte 0 Bit 0 Bit 1 corresponds to Byte 0 Bit 1 ..... Bit 8 corresponds to Byte 1 Bit 0 etc. Please note that the way the bytes will be actually sent on the bus does not impact this representation: they will always be seen by the software as a byte array. Note also that the absolute position of the signal in the PDU is then determined by the definition of the packingByteOrder attribute of the signal.
systemSignal	SystemSignal	1	reference	Reference to the system signal that shall be transported at this position in the frame.

The following example is supposed to explain the attribute `packingByteOrder` in more detail. The `packingByteOrder` attribute defines the way byte frontiers are crossed when mapping data elements to frames. The example shows how a nine bit data element fills a PDU bit by bit (starting from signal bit 0). It starts somewhere in Byte n and if the end of the byte is reached, there are two choices to continue.
The two options are to go ahead from byte n to byte n+1 (“Little Endian”) or to go backwards from byte n to byte n – 1 (“Big Endian”). The startPosition is now defined

depending on the endianess: if little endian, the position of the data element is bit 0; if big endian, the position of the data element is bit 9.

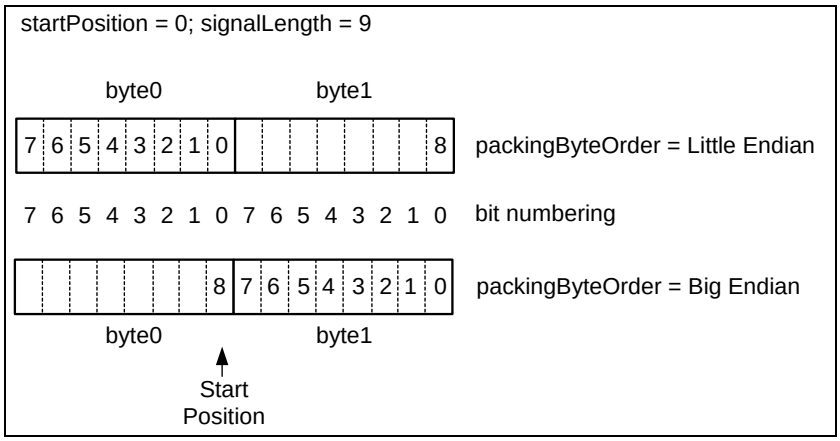

Figure 19: PackingByteOrder Example

5.2.2 Multiplexer

Multiplexing is used to transport varying signals at the same position of a single PDU. A multiplexed PDU consists of a static part and a dynamic part, where the static part consists of zero or more signals and the dynamic part consists of the selector field and one or more signals.
The layout of the signals in the PDU is defined by the SubPDU element. In the dynamic part a SubPDU is a sequence of SystemSignals whose appearance depends on a certain value of a multiplexers switch. In the static part the content of the SubPDU is the same regardless of the selector field.
It is possible that the dynamic part consist of more than one sub-part. The Multiplexer element describes the position of a selector within a PDU. A selector is a bitfield of certain length, by the value of which the corresponding data region of the subPDU must be interpreted dynamically, i.e. at run-time.
The multiplexer definition consists of the following elements:

Class	**Multiplexer**			
Package	M2::AUTOSARTemplates::SystemTemplate::PDUStructure			
Class Description	A multiplexer is used to define variable parts within a PDU that may carry different signals. The receivers of such a PDU can determine which signals are transmitted by evaluating the selector field, which carries a unique selector code for each sub-part.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
Data	SubPDU	0..*	aggregation	According to the value of the selector field some parts of the PDU have a different layout. These parts of the PDU that can contain different signals are called dynamic part.
selectorFieldByteOrder	ByteOrderEnum	1	aggregation	This parameter defines the byte order of the selector field. The byte ordering "Little Endian" (MostSignificantByteLast) and "Big Endian" (MostSignificantByteFirst) can be selected. The value of this attribute impacts the absolute position of the selector field into the PDU.
selectorFieldLength	Integer	1	aggregation	The size in bits of the selector field shall be configurable in a range of one bit and eight bits.
selectorFieldStartPosition	Integer	1	aggregation	This parameter is necessary to describe the position of the selector field within the PDU. It denotes the least significant bit for "Little Endian" and the most significant bit for "Big Endian" (see the description of the selectorFieldByteOrder attribute).

Class	**SubPDU**			
Package	M2::AUTOSARTemplates::SystemTemplate::PDUStructure			
Class Description	A multiplexed PDU consists of a static part and a dynamic part, where the static part consists of zero or more signals and the dynamic part consists of the selector field and one or more signals. Each part of a multiplexed PDU, the static part and the different dynamic parts, are described as a SubPDU.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
dynamicSubPDUTiming	Timing	0..*	aggregation	This relationship is only valid for the dynamic part of the PDU. This relationship allows to define a timing for a SubPdu with a certain selectorFieldCode.
includedSignal	SignalPosition	0..*	aggregation	Definition of the signals included in the SubPDU with their position. All signals included must have the same transmitter (as specified in the cluster signal).
selectorFieldCode	Integer	0..1	aggregation	The selector field is part of a multiplexed PDU. It consists of contiguous bits. The value of the selector field selects the layout of the multiplexed part of the PDU. This attribute is only valid for the dynamic part of the PDU.
startPosition	Integer	1	aggregation	The position of the static and the dynamic part of the multiplexer in the PDU.
SubPDUSize	Integer	1	aggregation	The size of the SubPDU in bits. The size is limited by the PDUSize.

5.3 Signal Groups

To support the AUTOSAR concept of complex data types the AUTOSAR COM layer provides signal groups.
Every record or array element of a complex data type requires a `SystemSignal` for the transmission. But the RTE has to guarantee the atomic transmission of data. A signal group shall be transmitted and received atomically; therefore it provides data consistency for complex data types.
A `SystemSignalGroup` refers to a set of `SystemSignals` that must always be kept together in a common Frame.

Class	**SystemSignalGroup**			
Package	M2::AUTOSARTemplates::SystemTemplate::SystemSignal			
Class Description	A signal group refers to a set of signals that must always be kept together. A signal group is used to guarantee the atomic transfer of AUTOSAR composite data types.			
Base Class(es)	ARObject, Identifiable, PackageableElement, NonSplitableElement, ARElement			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
containedSignals	SystemSignal	1..*	reference	Reference to a set of signals that must always be kept together.

5.4 Communication Matrix

The purpose of this chapter is to specify and describe the communication matrices. In order to do that, the characteristics of the frames must be considered. It was the objective to find a communication description, which considers all communication clusters/buses used in the automotive industry. However, the description should enable also a simple adaptation to future buses and to other here not considered communication links.
To reach the objective, general parameters are defined. These general parameters describe all possible communication links. However, every single communication cluster has its own characteristics, which do not apply to the other communication clusters. These characteristics are considered as communication cluster specific parameters. The description of the communication cluster specific parameters enables a detailed description of communication.

Class	**CommunicationMatrixInstance**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Class Description	A communication matrix defines frames that are transported on a specific communication system, with timing etc. The CommunicationMatrixInstance references to the communicationMatrixType that shall be used in the system.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
communicationMatrixType	CommunicationMatrixType	1	reference to type	References to the communication matrix type used for this instance.

Class	**CommunicationMatrixType**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	A communication matrix defines frames that are transported on a specific communication system, with timing etc. The communication matrix is separated from the topology, since the same topology may be used with different communication matrices. The same communication matrix type may be reused in different systems. Reuse is however only possible if the systems reuse the same topology (i.e. system topology type) as well.			
Base Class(es)	ARObject, Identifiable, PackageableElement, NonSplitableElement, ARElement			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
communicationCluster	CommunicationMatrixType_communication-Cluster	1	aggregation	Reference to the communication cluster this communication matrix is defined for. In each actual system, there is a 1-to-1 relation between communication clusters and communication matrices. But since a topology may be reused in different systems with different communication matrices, there may be several communication matrices referencing to the same communication cluster.
frame	Frame	0..*	aggregation	The frames that are defined in the communication matrix
linSchedulingTable	LinSchedulingTable	0..*	aggregation	The master task transmits frame headers based on a schedule table. The master application may use different

				schedule tables and select among them.
systemTopologyType	SystemTopologyType	1	reference	Each communication matrix is defined for exactly one topology. The same communication matrix type may be reused in different systems. But this is only possible if the systems reuse the same topology as well.

Usage of the Template to describe the “System Constraints”

The System Constraint Description describes all frames that are predefined on all communication clusters of a vehicle. The predefinition of the communication matrix or parts of the communication matrix force the system generator to use the given Frame-Structure, Frame-ID, Frame-Timing etc. of the assigned signals. Constraints for the system generator arise here e.g. from the used bus bandwidth, used identifiers as well as from the timing and at which position in a frame a signal is transmitted on the channel.
Such a manual definition of the communication can be made for any reason where it is necessary to restrict the system generator. They are necessary for example due to integration or compatibility reasons. One example is the usage of legacy ECUs in an AUTOSAR System. The frames that are transmitted or received by these legacy ECUs are constraints for the system generator because they cannot be changed, if the compatibility is supposed to be achieved without any changes at the legacy ECUs.

Usage of the Template to describe the “System Configuration”

In contrary to the System Constraint Description the final System Configuration Description contains all signals and frames that will be sent by any ECU in the car. No matter if they were predefined (system constraint) or if they were generated by the system generator. The available information, in addition to the information, which is inserted by the AUTOSAR ECU configuration generator step, will be used as input to configure the Basic SW for the communication e.g. bus driver and the COM-Layer.

5.4.1 General description of the communication matrix

Due to the very different characteristics of frames of the individual communication clusters it is necessary to examine which parameters of the frames are communication cluster specific and which ones are general. To achieve the objective

the buses LIN, CAN and FlexRay were considered. MOST is outside the scope of AUTOSAR Phase 1.
These buses are the most used and standardized buses for the inter ECU communication in vehicles. Therefore a limitation to these buses occurs in the first step (see also main requirement AUTOSAR Main160, see Requirements VFB Spec). However, the goal is to find a description for the communication, which is as universal as possible. This chapter describes the parts of the metamodel, which are valid for more than one of the examined data buses. The description of the bus specific parts in the metamodel will follow in the next chapter.

5.4.1.1 Frame

The `Frame` describes a data frame on a communication cluster. Each `Frame` has a relationship to exactly one physical channel. Over this physical channel the `Frame` will be transmitted. If a frame is sent over several physical channels (e.g. in case of redundancy), several frame instances have to be defined, one for each physical channel. A `Frame` can be identified unambiguously on a physical channel by the `frameID`.
Furthermore it is possible to describe through which communication port the data frame will be transmitted and received. Several receivers of the `Frame` can exist. But only one transmitter can be defined. The timing with which the frame is transmitted is described by the `Timing` element.
A Frame can be composed of several PDUPrototypes. For example, it might be possible that one FlexRay Frame contains more than one PDU. The signal layout is defined in the `PDUType`. More details can be found in chapter 5.2.

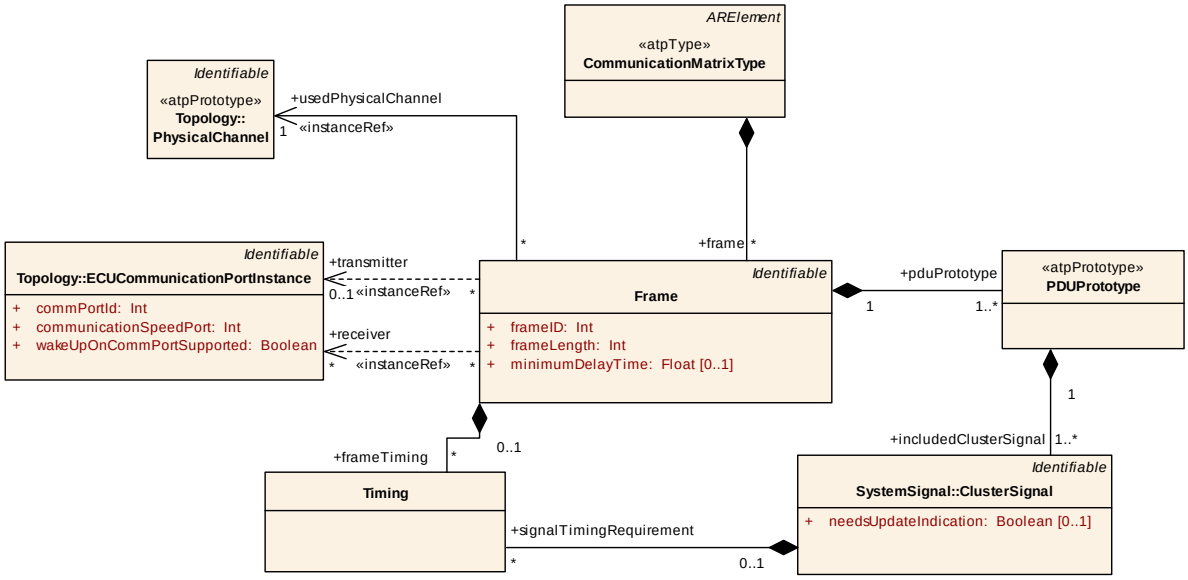

Figure 20: Frame definition in the metamodel (FrameInstance)

Class	**Frame**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	Data frame which is sent over a communication medium. Each Frame can be identified per channel by an Identifier (ID).			
Base Class(es)	ARObject, Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
frameID	Integer	0..1	aggregation	The Identifier (ID) of the data frame on the channel. In FlexRay each frame is identified by its slot id and communication cycle (attributes of AbsoluetlyScheduledTiming).
frameLength	Integer	1	aggregation	The frameLength specifies the length of the payload section in bytes.
frameTiming	Timing	0..*	aggregation	One frame can have 0..* Timings, since it is possible that e.g. a frame is sent cyclic, but at an occurred event it is sent EventControlled. Furthermore, at different ECU states or signal conditions, a different timing can be valid. A multiplicity of zero means that no timing is specified. If no timing is specified the frame is defined (i.e. the ID is allocated) but will not be transmitted on the system. (e.g. useful for placeholder frames).
minimumDelayTime	Float	0..1	aggregation	A minimum delay time between transmissions in seconds. If a transmission is requested before the minimumDelayTime expires, the next transmission is postponed until the delay time expires. The minimum delay time for the next transmission starts the moment the previous transmission is confirmed. (OSEK COM)
pduPrototype	PDUPrototype	1..*	aggregation	A Frame can be composed of several PDUs (e.g. FlexRay).
receiver	Frame_receiver	0..*	reference to instance	ECUs which receive the frame. This can also be a gateway. An ECU can contain more than one communication controller to a data bus. Therefore, it is important to describe over which controller the frame will be received. Several receivers of the

				information can exist. The set of receivers defined for the frame must be a superset of all receivers specified in any cluster signal that is included in the frame. In other words, if one of the cluster signals included in the frame specifies a certain receiver, this receiver must be included for the corresponding frame as well.
transmitter	Frame_transmitter	0..1	reference to instance	The transmitter describes the ECU which sends the frame. An ECU can contain more than one communication controller to a data bus. Therefore, it is important to describe over which controller the frame will be sent. The transmitter defined for the frame must match with the transmitter defined for all cluster signals that are included in the frame.
usedPhysicalChannel	Frame_usedPhysicalChannel	1	reference to instance	Reference to the physical channel used to transport this frame. If a frame is sent over several physical channels (e.g. in case of redundancy), several Frames have to be defined in the System Description, one for each physical channel.

5.4.1.2 Timing

`Timing` defines the time behavior of `Frames` and is also used to describe timing requirements on signal level.
The description of the Timing must be precise enough that the System Generator can calculate the bus load and the resulting time for the transmission of a frame. Another aim is to get a general view when and how often in the normal case frames are transmitted to have the possibility to configure the COM Modules of the ECU e.g. the filters.
This section describes different types of timings, with which the frame can be transmitted. All these timing types are modeled as a specialization of the meta-class `Timing`.

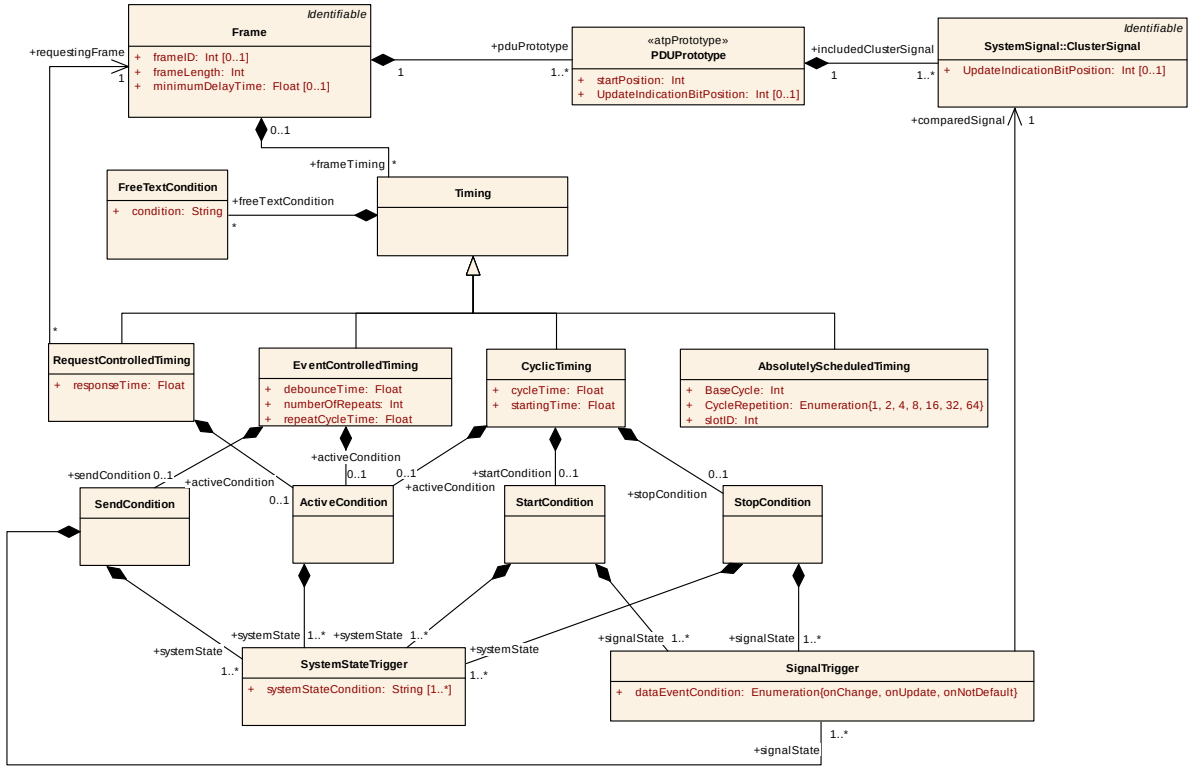

Figure 21: Timing Description (FrameTiming)

Class	**Timing**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	A timing defines time behavior (e.g. Cyclic, Event Triggered) of a frame, PDU or a Signal.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
freeTextCondition	FreeTextCondition	0..*	aggregation	If a Condition is aggregated in a Timing, then the timing is only valid if the condition holds.

Class	**RequestControlledTiming**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	Specification of a request driven sending behavior, e.g. used on CAN buses. Semantics of this communication mechanism is that basic software stores values but does not send it out until a frame requesting the information is received.			
Base Class(es)	ARObject, Timing			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
activeCondition	ActiveCondition	0..1	aggregation	If an active condition is aggregated in a RequestControlledTiming, then the request will be registered if the condition holds.
requestingFrame	Frame	1	reference	Reference to the requesting frame. This frame must be received, before the value can be send out.
responseTime	Float	1	aggregation	Specification of the time in seconds that is needed before the frame can be sent after the requests arrival.

Class	**EventControlledTiming**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	Specification of a event driven sending behavior, e.g. used on CAN buses. The Frame is sent "numberOfRepeats" times separated by the repeatCycleTime. If numberOfRepeats = 1, then the frame is sent just once. The debounce Time must elapses before the frame can be sent the next time.			
Base Class(es)	ARObject, Timing			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
activeCondition	ActiveCondition	0..1	aggregation	If an active condition is aggregated in a EventControlledTiming, then the event will be registered if the condition holds.
debounceTime	Float	1	aggregation	Specification of the time in seconds that elapses before the frame can be sent the next time (Minimum repeat gap between two frames). This time is different to the minimumDelayTime, defined on the Frame Level. If a transmission is requested before the debounceTime expires, this request will be ignored. The next transmission will not be postponed.
numberOfRepeats	Integer	1	aggregation	Number of times the frame is sent if the condition is met
repeatCycleTime	Float	1	aggregation	Specification of the repeating cycle in seconds that is used to repeat the frame.
sendCondition	SendCondition	0..1	aggregation	If a Send Condition is aggregated in a EventControlledTiming, then the frame will be sent if the condition holds.

Class	**CyclicTiming**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	Specification of a cyclic sending behavior, e.g. used on CAN buses.			
Base Class(es)	ARObject, Timing			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
activeCondition	ActiveCondition	0..1	aggregation	If an active condition is aggregated in a Cyclic Timing, then the cycle can be started if the condition holds.
cycleTime	Float	1	aggregation	Specification of the repeating cycle in seconds whenever the frame described by this timing is sent.
startCondition	StartCondition	0..1	aggregation	If a StartCondition is aggregated in a Cyclic Timing, then the cycle is only started if the condition holds.
startingTime	Float	1	aggregation	Specification of the time in seconds that is needed before the frame can be sent the first time. This starting time is relative to the StartCondition.
stopCondition	StopCondition	0..1	aggregation	If a Stop Condition is aggregated in a Cyclic Timing, then the cycle is only stopped if the condition holds.

Class	**AbsolutelyScheduledTiming**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	Each frame in FlexRay is identified by its slot id and communication cycle. A description is provided by the usage of AbsolutelyScheduledTiming. In the static segment a frame can be sent multiple times within one communication cycle. For describing this case multiple AbsolutelyScheduledTimings have to be used. The main use case would be that a frame is sent twice within one communication cycle.			
Base Class(es)	ARObject, Timing			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
BaseCycle	Integer	1	aggregation	The first communication cycle where the frame is sent
CycleRepetition	AbsolutelyScheduledTiming-CycleRepetitionEnum	1	aggregation	The number of communication cycles (after the first cycle) whenever the frame is sent again. The FlexRay communication controller allows only determined values.
slotID	Integer	1	aggregation	In the static part the SlotID defines the slot in which the frame is transmitted. The SlotID also determines, in combination with FlexrayCluster::numberOfStaticSlots, whether the frame is sent in static or dynamic segment. In the dynamic part, the slot id is equivalent to a priority. Lower dynamic slot ids are all sent until the end of the dynamic segment. Higher numbers, which were ignored that time, have to wait one cycle and then must try again.

The validity of timing can depend on a defined condition. If a condition is aggregated to a timing, then the timing is only valid if the condition holds. This way it is possible e.g. to define for every frame in which general vehicle state this frame is transmitted e.g. terminal_15, terminal_30. If such conditions for the frame transmission are described, it is for example possible to make busload estimation depending on such vehicle states. So you get a more precise picture of the busload and for example a better estimation for the worst case busload because not all frames are sent in every vehicle state.
The description of the possible conditions is taken over from FIBEX [13]. The following timing conditions can be described:

Class	**FreeTextCondition**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	A Condition can be formulated as a free text. If a FreeTextCondition is aggregated in a Timing, then the timing is only valid if the condition holds.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
condition	String	1	aggregation	Condition formulated as a free text.

Class	**StopCondition**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	Condition that must be fulfilled by the system or some signals that the cycle is stopped.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
signalState	SignalTrigger	1..*	aggregation	Reference to the condition that must be fulfilled by some signals that the cycle is stopped.
systemState	SystemStateTrigger	1..*	aggregation	Reference to the system state that must rule that the referring timing is activated. Use one trigger with many systemStateConditions, if these states must rule simultaneously for activation. Use a separate trigger for each state if only one state needs to rule for activation.

Class	**StartCondition**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	Condition that must be fulfilled by the system or some signals that the cycle is started.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
signalState	SignalTrigger	1..*	aggregation	Reference to the condition that must be fulfilled by some signals that the cycle is started.
systemState	SystemStateTrigger	1..*	aggregation	Reference to the system state that must rule that the referring timing is activated. Use one trigger with many systemStateConditions, if these states must rule simultaneously for activation. Use a separate trigger for each state if only one state needs to rule for activation.

Class	ActiveCondition			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	Precondition that must be fulfilled by the system. If this precondition is not fulfilled the timing that aggregates this ActiveCondition is not active. If a timing is not active, the frame is either not sent at all (if no other active timing exists), or the frame is sent since another timing is active and all conditions are met in that timing.			
Base Class(es)	ARObject			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
systemState	SystemStateTrigger	1..*	aggregation	Reference to the system state that must rule that the referring timing is activated. Use one trigger with many systemStateConditions, if these states must rule simultaneously for activation. Use a separate trigger for each state if only one state needs to rule for activation.

Class	SendCondition			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	Condition (Event) that must be fulfilled to cause the frame to be sent.			
Base Class(es)	ARObject			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
signalState	SignalTrigger	1..*	aggregation	Reference to the signal state that must rule that the referring timing is executed.
systemState	SystemStateTrigger	1..*	aggregation	Reference to the system state that must rule that the referring timing is activated. Use one trigger with many systemStateConditions, if these states must rule simultaneously for activation. Use a separate trigger for each state if only one state needs to rule for activation.

Class	**SystemStateTrigger**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	System state that must rule that the referring timing is activated.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
systemStateCondition	String	1..*	aggregation	This attribute describes the system state. Examples for values are: CLAMP_15, CLAMP_30, CLAMP_87, CLAMP_RADIO, CHANNEL_ACTIVE. (These values are only examples, they are not standardized by AUTOSAR)

Class	**SignalTrigger**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	The Signal Trigger describes the signal state that must rule that the referring timing is executed. The attribute dataEventCondition defines the primary condition when the condition is fulfilled. It is a Boolean Condition. If this Condition is TRUE, then the referring timing can be executed.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
comparedSignal	ClusterSignal	1	reference	reference to the signal which is compared to the conditionValue in the SignalCondition.
dataEventCondition	SignalTriggerDataEventCondition-Enum	1	aggregation	Specifies what operation on the data mapped to this signal will trigger the sending of a frame. onChange: send frame if data written changed compared to last value written onUpdate: send frame whenever data is written onNotDefault: send frame whenever value is different from default value

5.4.1.3 PDUPrototype

The PDU describes a collection of signals for transfer over a communication cluster. It is possible that one frame contains more than one PDU (e.g. FlexRay). For this reason the `Frame` can contain several `PDUPrototypes`.

Class	**PDUPrototype**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	The PDU (Protocol Data Unit) is a collection of signals for transfer between nodes in a network. The PDU is sent within a frame over a communication medium. A PDU can contain multiplexed data.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
includedClusterSignal	ClusterSignal	1..*	aggregation	Definition of clusterSignals included in the PDU.
pduType	PDUType	1	reference to type	Reference to the PDU description that defines the structure of the PDU. Different PDUs can have the same structure, for example they can have the same size and include the same signal layout.
startPosition	Integer	1	aggregation	This parameter is necessary to describe the position of a PDU within a frame.
UpdateIndicationBitPosition	Integer	0..1	aggregation	Indication to the receivers that the corresponding PDU was updated by the sender. The AUTOSAR FlexRay Interface explicitly defines a location for the update bit in the Frame that aggregates this PDUPrototype.

5.4.2 Bus specific description of the communication matrix

Frames look different in the different bus technologies. For this reason a specialization is used, through which the bus specific attributes for the individual data buses can be specified. In this chapter the special characteristics of the data buses are considered.

5.4.2.1 CAN

In the following, the parameters will be specified, which are necessary to describe the CAN Frames.

General remark:

The frame identifier establishes also the priority of the frame. The lower the binary value, the higher is the priority.

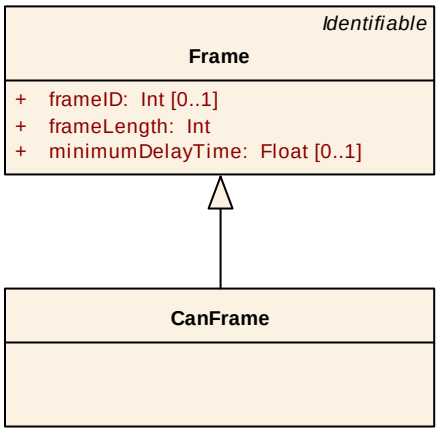

Figure 22: The general Frame attributes. (CANFrames)

Class	**CanFrame**
Package	M2::AUTOSARTemplates::SystemTemplate::Communication
Class Description	The frame description for the CAN Bus.
Base Class(es)	ARObject, Identifiable, Frame

5.4.2.2 FlexRay

In the following, the parameters will be specified, which are necessary to describe the FlexRay Frames. The parameters are based on the FlexRay specification 2.0.

FlexRay static channel parameters:

In the static segment all communication slots are of identical, statically configured duration and all frames are of identical, statically configured length.

The sending behavior where the exact time for the frames transmission is guaranteed must be specified. Thus, the communication cycle where the frame is sent and the slot in which the frame should be transmitted must be described. A slotID is used no more than once on each channel in a communication cycle. The valid range is 1..2047.

In the cycle counter field of every frame, the current value of the cycle counter is transmitted (see FlexRay frame format). This value is incremented at the beginning of each new cycle, ranging from 0 to 63, and is reset to 0 after a sequence of 64 cycles. The value of the cycle counter can be used for frame transmission and reception filtering. Cycle multiplexing means that we use the cycle counter for the frame transmission and reception. This feature gives the possibility for one ECU to send (and receive) different frames with different payload data and different update cycles (signal period) in the same slot. The counting of the cycle counter occurs through the FlexRay-Protocol (the Bus-Controller). In a communication cycle the cycle is equal for all frames.

In the static segment frames can be sent multiple times within one communication cycle. For describing this case multiple `AbsolueltyScheduledTimings` have to be used.

The AbsolueltyScheduledTiming element contains the attributes which are necessary to describe the slot and the cycle counter. The cycle counter is described by the two attributes baseCycle and cycleRepetition. More details can be found in chapter 5.4.1.2.

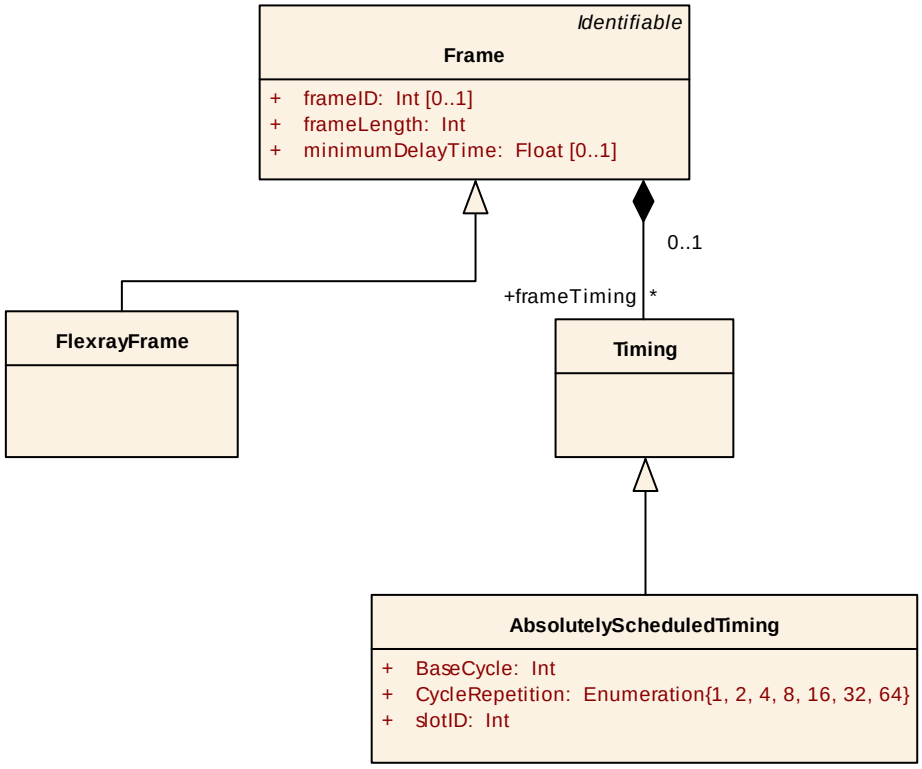

Figure 23: Frame description in FlexRay (FlexRayFrames)

Class	**FlexrayFrame**
Package	M2::AUTOSARTemplates::SystemTemplate::Communication
Class Description	Unlike with other communication systems - as for example CAN - the Frame ID is not sufficient for the identification. On FlexRay the communication cycle is used with SlotID for frame identification. These attributes are described by the AbsolutelyScheduledTiming class.
Base Class(es)	ARObject, Identifiable, Frame

FlexRay dynamic channel parameters:

In the dynamic segment the duration of communication slots may vary in order to accommodate frames of varying length.

Furthermore, in the dynamic part, the slot id is equivalent to a priority. The higher the number the lower is the priority.

But the frames in the static and in the dynamic channel have the same format. Each FlexRay Frame is identified by its slot id and communication cycle. A description is provided by the usage of AbsolueltyScheduledTiming.

If the behavior of a frame is cyclic or event triggered, many timings can be used together. In addition to the `AbsolueltyScheduledTiming` the `EventControlledTiming` or the `CyclicControlledTiming` can be assigned to a frame.

5.4.2.3 LIN

In the following, the parameters will be specified, which are necessary to describe the LIN Frames. The parameters are based on the LIN specification 2.0. In order to describe the LIN Communication the LIN Scheduling Table element is defined. The LIN Scheduling Table is enclosed directly by the Communication Matrix Element. The Scheduling Table consists of table entries, which contain Frame Slots.

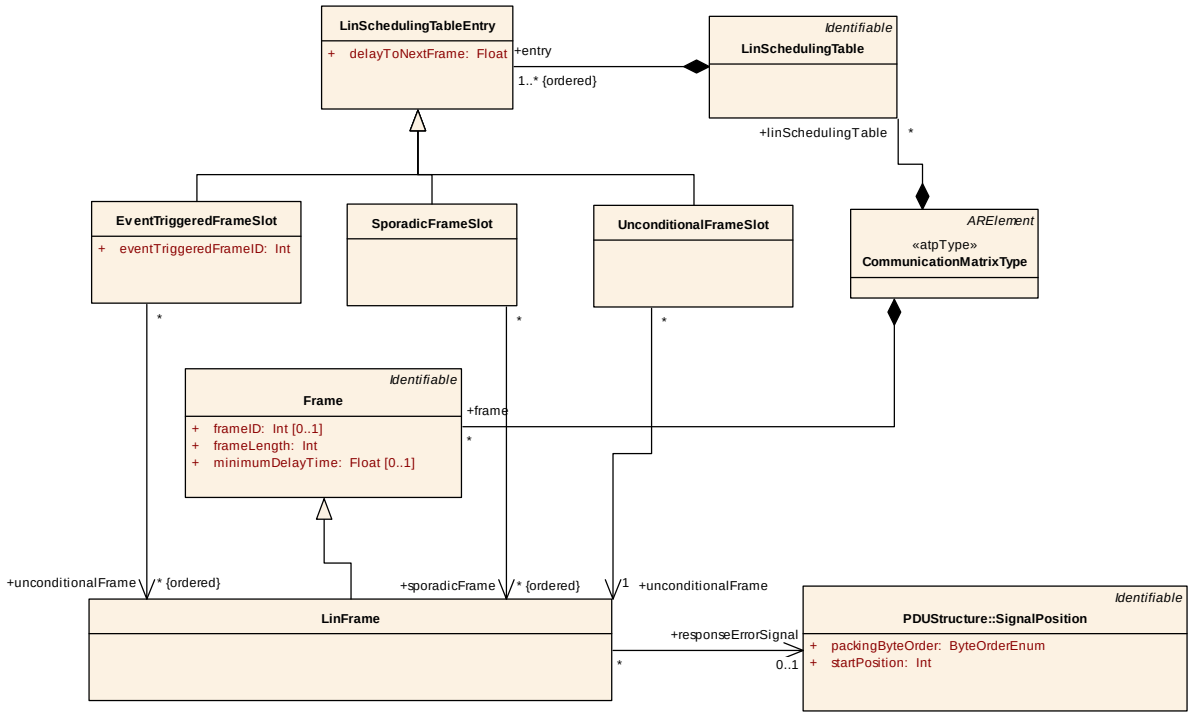

Figure 24: Frame Description in LIN (LINFrames)

Class	**LinFrame**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	The frame description for the LIN Bus.			
Base Class(es)	ARObject, Identifiable, Frame			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
responseErrorSignal	SignalPosition	0..1	reference	This reference denotes the signal within this frame (if present) that is used to implement Response_Error. If no such signal exists in this frame, the reference is omitted. LIN 2.0 Spec states: "Each slave shall send one status bit signal, Response_Error, to the master node in one of its transmitted frames. Response_Error shall be set whenever a frame received by the node or a frame transmitted by the node contains an error in the response field. Response_Error shall be cleared after transmission."

Class	**LinSchedulingTable**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	The master task (in the master node) transmits frame headers based on a schedule table. The schedule table specifies the identifiers for each header and the interval between the start of a frame and the start of the following frame.			
Base Class(es)	ARObject, Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
entry	LinSchedulingTableEntry	1..*	aggregation	The position of a table entry in the scheduling table is described by the "isOrdered" attribute.

Class	**LinSchedulingTableEntry**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description				
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
delayToNextFrame	Float	1	aggregation	Relative delay of the frame in seconds to its successor in the schedule table.

Class	**EventTriggeredFrameSlot**
Package	M2::AUTOSARTemplates::SystemTemplate::Communication
Class Description	Frame slot for an event triggered frame. LIN Spec 2.0 defines: "An event triggered frame is used as a placeholder to allow multiple slave nodes to provide its response. This is useful when the signals involved are changed infrequently." The event triggered frame has an own frame id that is sent by the LIN master. The slaves may answer with one of the unconditional frames referenced. LIN Spec 2.0 defines: "If more than one unconditional frame is associated with one event triggered frame (which is the normal case) they shall all be of equal length, use the same checksum model (i.e. mixing LIN 1.3 and LIN 2.0 frames is incorrect) and, furthermore, they shall all be published by different slave tasks."
Base Class(es)	ARObject, LinSchedulingTableEntry

Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
eventTriggeredFrameID	Integer	1	aggregation	
unconditionalFrame	LinFrame	0..*	reference	Reference to the unconditional frame(s) associated with the event triggered frame. From LIN 2.0 Specification: "If more than one unconditional frame is associated with one event triggered frame (which is the normal case) they shall all be of equal length, use the same checksum model (i.e. mixing LIN 1.3 and LIN 2.0 frames is incorrect) and, furthermore, they shall all be published by different slave tasks." The unconditional frames are ordered in decreasing priority. In the System Templates, this means that all LinFrames referenced need to furthermore define attribute eventTriggeredFrameID, set to the same value.

Class	**SporadicFrameSlot**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	By use of sporadic frames, the LIN master may send different unconditional frames in a specific scheduling table position, depending on events that occurred. From LIN 2.0 specification: "If multiple sporadic frames are associated with the same frame slot (the normal case), the most prioritized of the sporadic frames (which has an updated signal) shall be transferred in the frame slot. If none of the sporadic frames associated with a frame slot has an updated signal the frame slot shall be silent."			
Base Class(es)	ARObject, LinSchedulingTableEntry			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
sporadicFrame	LinFrame	0..*	reference	List of the unconditional frames that may be sent in this frame slot, ordered in decreasing priority.

Class	**UnconditionalFrameSlot**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	Frame slot for an unconditional frame. An unconditional frame is a signal-carrying frame that is always sent in its allocated frame slot.			
Base Class(es)	ARObject, LinSchedulingTableEntry			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
unconditionalFrame	LinFrame	1	reference	Reference to the unconditional frame sent in the unconditional frame slot.

Communication Path:
All signals/frames, which have to be published in a LIN network, are triggered by the master. The master sends out a frame-ID and a slave node or the master itself response. Each network node has to decide whether the published information is needed for its application. The meta-class `LINCommPortInstance` contains the attribute `Role`. With this attribute it is possible to assign a slave or master role to a communication port. Thus, the LIN communication path is described in the metamodel.

5.5 Gateways

A gateway is a function within an ECU that performs as a PDU or signal mapping function between two or more communication clusters. The system-generator generates the necessary information for the gateway tables (logical gateway table) according to mapping of the Software Components. From this logical gateway table a generator can generate the binary format of the gateway table either for an AUTOSAR ECU or an OEM specific gateway-table for a NON-AUTOSAR ECU. In the template there exists for each Gateway ECU one gateway table that describes the relations between the connected channels by gateway entries.
One gateway entry in the System Template describes which signal or PDU is copied from one channel to another channel. The information between which of the connected channels the signal or PDU is copied arises from the references of the gateway entry. These references show the source and the destination `PDUPrototype`. Each `PDUPrototype` is contained by a frame. And this frame references to the according channel. The `ClusterSignal` has a direct reference to the `ClusterSignalTransmitter` and `ClusterSignalReceiver`.

Usage of the Template to describe the “System Constraints”
The System Constraint Description describes all gateways in the system including their gateway entries that are predefined. The predefinition of the gateways or parts of the gateways can be used to define manually the copying of signals or PDUs.
The reasons for such predefinitions are quite the same as for the predefinitions of the frames.
Usage of the Template to describe the “System Configuration”
In contrary to the System Constraint Description the final System Configuration Description describes all gateways with all their gateway entries. No matter if they were predefined (System Constraint) or if they were generated by the System Generator.

The information of this description will be used as input to configure the gateway Basic SW i.e. to create the binary gateway tables for the gateway ECUs.

5.5.1 Gateway description

The gateway element contains one or many Gateway Entries. In case that a gateway connects the two buses A and B, a gateway entry (`PDUGatewayEntry`) describes which `PDUPrototype` on channel A is mapped to which `PDUPrototype` on channel B. Both `PDUPrototypes` must be defined within a frame.
Another possibility is that a gateway entry (`SignalGatewayEntry`) describes which `ClusterSignal` on channel A is mapped to which `ClusterSignal` on channel B. The `ClusterSignal` must be described in a source and a destination PDU.
The gateway is defined as follows:

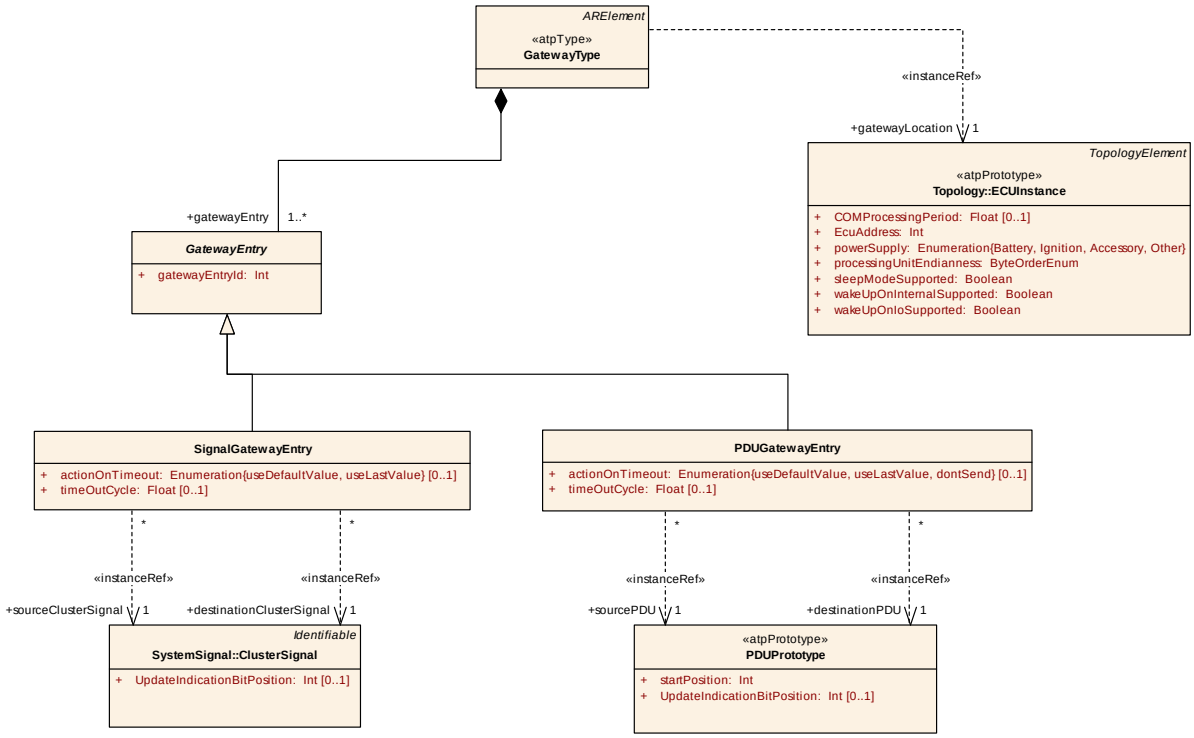

Figure 25: Gateway Description (Gateway)

Class	GatewayInstance			
Package	M2::AUTOSARTemplates::SystemTemplate			
Class Description	A gateway is a functionality within an ECU that performs a frame or signal mapping function between two or more communication clusters. The GatewayInstance references to the GatewayType used in the system.			
Base Class(es)	ARObject			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
gateway	GatewayType	1	reference to type	References to the gateway table type used for this instance. A gateway can be reused between different systems if all the communication matrices referenced implicitly (through the references to frames in the gateway entries) are reused as well.

Class	GatewayType			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	A gateway is a functionality within an ECU that performs as a PDU or signal mapping function between two or more communication clusters. The same GatewayType may be reused in different systems. This is however only possible if the system topology and communication matrices that are referenced from the gateway are reused as well.			
Base Class(es)	ARObject, Identifiable, PackageableElement, NonSplitableElement, ARElement			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
gatewayEntry	GatewayEntry	1..*	aggregation	Each Gateway consists of 1..* Gateway Entries.
gatewayLocation	GatewayType_gatewayLocation	1	reference to instance	

Class	GatewayEntry {abstract}			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	A GatewayEntry describes the information routing by references to the source and the destination PDUPrototype or ClusterSignal. There are two possible methods to route the data from one channel to another: SignalCopy and PDUCopy.			
Base Class(es)	ARObject			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
gatewayEntryId	Integer	1	aggregation	Identify the gateway entry in the gateway table. May be used to define the ordering of the entries.

Class	**SignalGatewayEntry**
Package	M2::AUTOSARTemplates::SystemTemplate::Communication
Class Description	SignalGatewayEntry describes the SignalCopy Method. In this case only one signal of the PDU will be copied to another PDU on the destination channel. The parameters on the destination channel like signal position will be defined through the PDU description of the destination PDU. I.e. the signal must be described in the source and the destination PDU. For each signal which should be copied, one entry in this table has to be created. If you want e.g. to create one PDU on the destination channel from two Signals of the source channel, you have to make two entries in the table. The Signal Gateway Trigger Condition is described by the usage of the SendCondition on an EventControlledFrame and the referring signal state.
Base Class(es)	ARObject, GatewayEntry

Attribute	*Datatype*	*Mul.*	*Link Type*	*Attribute Description*
actionOnTimeout	SignalGatewayEntryActionOn-TimeoutEnum	0..1	aggregation	What to do if a timeout occurred: - useDefaultValue: The default value of the signal will be transmitted. The default Value is the value that is sent in the frame when no actual or no valid value is available for the COM-Layer that could be sent. This/These values are defined in the SWC-T: InvalidValue; InitValue - useLastValue: The last sent value will be transmitted again.
destinationClusterSignal	SignalGatewayEntry_destination-ClusterSignal	1	aggregation	Reference to the signal on the destination channel.
sourceClusterSignal	SignalGatewayEntry_source-ClusterSignal	1	aggregation	Reference to the signal that is copied to the other channel.
timeOutCycle	Float	0..1	aggregation	Time in seconds. Is this value exceeded without that a value of the source is received, the action defined in actionOnTimeout is performed.

Class	**PDUGatewayEntry**			
Package	M2::AUTOSARTemplates::SystemTemplate::Communication			
Class Description	PDUGatewayEntry describes the PDUCopy Method. In this case the PDU will be copied, i.e. the destination PDU will have the same content (signals) and structure (position and length of signals) as the PDU on the source channel.			
Base Class(es)	ARObject, GatewayEntry			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
actionOnTimeout	PDUGatewayEntryActionOnTimeout-Enum	0..1	aggregation	What to do if a timeout occurred: - useDefaultValue: The default value of the signal will be transmitted. The default Value is the value that is sent in the PDU when no actual or no valid value is available for the COM-Layer that could be sent. This/These values are defined in the SWC-T: InvalidValue, InitValue - useLastValue: The last sent value will be transmitted again. - dontSend: nothing will be transmitted
destinationPDU	PDUGatewayEntry_destinationPDU	1	aggregation	Reference to the PDU on the destination channel.
sourcePDU	PDUGatewayEntry_sourcePDU	1	aggregation	Reference to the PDU that is copied to the other channel.
timeOutCycle	Float	0..1	aggregation	Time in seconds. Is this value exceeded without that a value of the source is received, the action defined in actionOnTimeout is performed.

5.6 Communication Protocols

Note: The following section explains how Communication Protocols can be defined using the System Template. As of AUTOSAR Release 2.1 there are no protocols standardized using this scheme. Based on these model elements however, Communication Protocols may be defined to be standardized and supported by the BSW modules in future versions of the standard.
In basic (traditional) sender receiver communication, data elements exchanged between software components are mapped to individual signals with matching size in frames on a bus. For example, a four bit integer value exchanged between software component A and software component B will be mapped to a four bit signal in specific CAN frame. However, there are cases where this simple one-to-one mapping is not the appropriate behavior, e.g.

- if the size of the data exchanged exceeds the frame size available on the bus
- if client-server communication with requests and responses that match a specific request is needed
- if acknowledgement and retransmission is needed to implement reliable communication
- etc.

In those cases, a communication protocol will be used. Since communication protocols may come in many different flavors, it is impossible to define a meta-model that allows for a precise definition of all aspects of the communication protocol, including timing, error handling etc. Instead, the meta-model restricts itself to describe

- which protocol is used for a specific communication
- which network resources are used by the protocol
- what VFB communication is mapped to a specific protocol instance

With this information, it is up to the basic software generators / integration engineer to derive how that specific communication protocol implementation must be configured to run properly.
The chapter starts with two example communication protocols that may be used in the AUTOSAR context. They serve as a base to understand better what kinds of protocols and mappings must be describable in the System Template.
Then the meta-model for communication protocols and mapping are explained. Finally, the examples are continued and it is shown how an application of the two communication protocols could be described.
Example: ISO 15765 Transport Protocol

As an example, a simple protocol that follows the frame format of the transport protocol defined in ISO 15765-2: “Road vehicles – Diagnostics on Controller Area Networks (CAN) – Part 2: Network Layer Services” will be used in this chapter.
We only look at the frame format and only at normal addressing mode. In normal addressing mode, the CAN identifier is used to identify source and target address as well as target address type (physical or functional) are defined by the CAN identifier.

Four different frame types are defined:
Single Frame (SF); is used to transmit unsegmented data from source to target. Target may be a physical address (unicast, single sender, single receiver) or functional address (multicast, single sender, multiple receivers).
First Frame (FF) and Consecutive Frames (CF) are used to transmit messages that do not fit into a single frame. Only physical addresses (unicast) are allowed for the target. Flow Control (FC) is used in the opposite direction to acknowledge frames and to regulate how fast the CF frames may be sent by the source.
The 8 bytes of a CAN frame are used as follows:

N_PDU type	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Single Frame	N_PCI	N_Data						
FirstFrame	N_PCI		N_Data					
ConsecutiveFrame	N_PCI	N_Data						
FlowControl	N_PCI			not present				

The N_PCI (protocol control information) in turn is split up into a common four bit identifier and a part that is PDU_type dependent:

N_PDU type	Byte 1		Byte 2	Byte 3
	bit 7..4	bit 3..0		
Single Frame	0000	DL		
FirstFrame	0001	DL		
ConsecutiveFrame	0010	SN		
FlowControl	0011	other info		

5.6.1 Example: A simple acknowledgement protocol for reliable transport

In this chapter a protocol is sketched that could be used to implement such a VFB communication. The chapter will not provide a full specification of the protocol, just the message formats to illustrate certain details what must be describable in the System Template.
In the previous chapter, ISO-TP was shortly described. In that protocol and in related protocols, always a full frame is reserved in each communication direction to implement the protocol. While this is useful and necessary if bigger amounts of data are transferred (and thus the amount of data transferred per frame should be maximized), there are cases where allocation of full frames is not the most efficient handling:
Assume a Software component A that wants to send two four-bit data elements DataA1 and DataA2 to Software component B and software component C. A, B and C are mapped to different ECUs that all connect to the same CAN bus. For some reasons A wants feedback from B and C whether the data was received properly. Assume furthermore that DataA1 has strict timing requirements and thus must be sent with high priority and DataA2 may be sent with low priority on the CAN bus.
Thus a protocol that imposes minimal resource requirements to the bus could look as follows:

- four bits in a frame FA1 sent from A with high priority for sending of DataA1. Both B and C receive and evaluate that frame.
- four bits in a frame sent FA2 from A with low priority for sending of DataA2. Both B and C receive and evaluate that frame.
- one bit in a frame FB1 sent from B (and FC1 for C) with high priority to acknowledge DataA1, only received and evaluated by A.
- one bit in a frame FB2 sent from B (and FC2 for C) with low priority to acknowledge DataA2, only received and evaluated by A.

With the following protocol: All frames are sent cyclic with the same cycle time. As long as B and C receive data in FA1 and FA2 properly, they set the respective acknowledgement bit in frames FB1 and FB2 (resp. FC1 and FC2) to ‘1’, otherwise they set it to ‘0’ to indicate that no data was received.
This protocol would fulfill the requirements imposed by the VFB communication mechanism and has resource requirements of 12 bits.
To implement the same behavior with an ISO-TP-like protocol, eight full frames would be needed, since acknowledge protocols require point-to-point connections:

- Frames FA2B1 for sending DataA1 from A to B and FB2A1 for the acknowledgement from B to A, both with high priority.
- Frames FA2C1 for sending DataA1 from A to C and FC2A1 for the acknowledgement from C to A, both with high priority.
- Four more frames accordingly with low priority, for sending and acknowledgement of DataA2.

This example shows that it is useful and necessary to allow protocols that do not use a whole frame but only some bits or bytes within a frame. This must be taken into account when modeling protocols in the System Template.

5.6.2 Communication Protocol Modeling

Before VFB communication can be mapped to a communication protocol, the protocol has to be defined. This definition consists of two parts:

- A general definition of the communication protocol, defined in `CommProtocolType` that contains all general information about the communication protocol, such as timing behavior etc. Several instances of the protocol will refer to the same type definition.
- The communication protocol instance that will be used for the specific VFB communication that shall be mapped. The Instance definition contains all information that is specific for the protocol instance and references to the protocol type description.

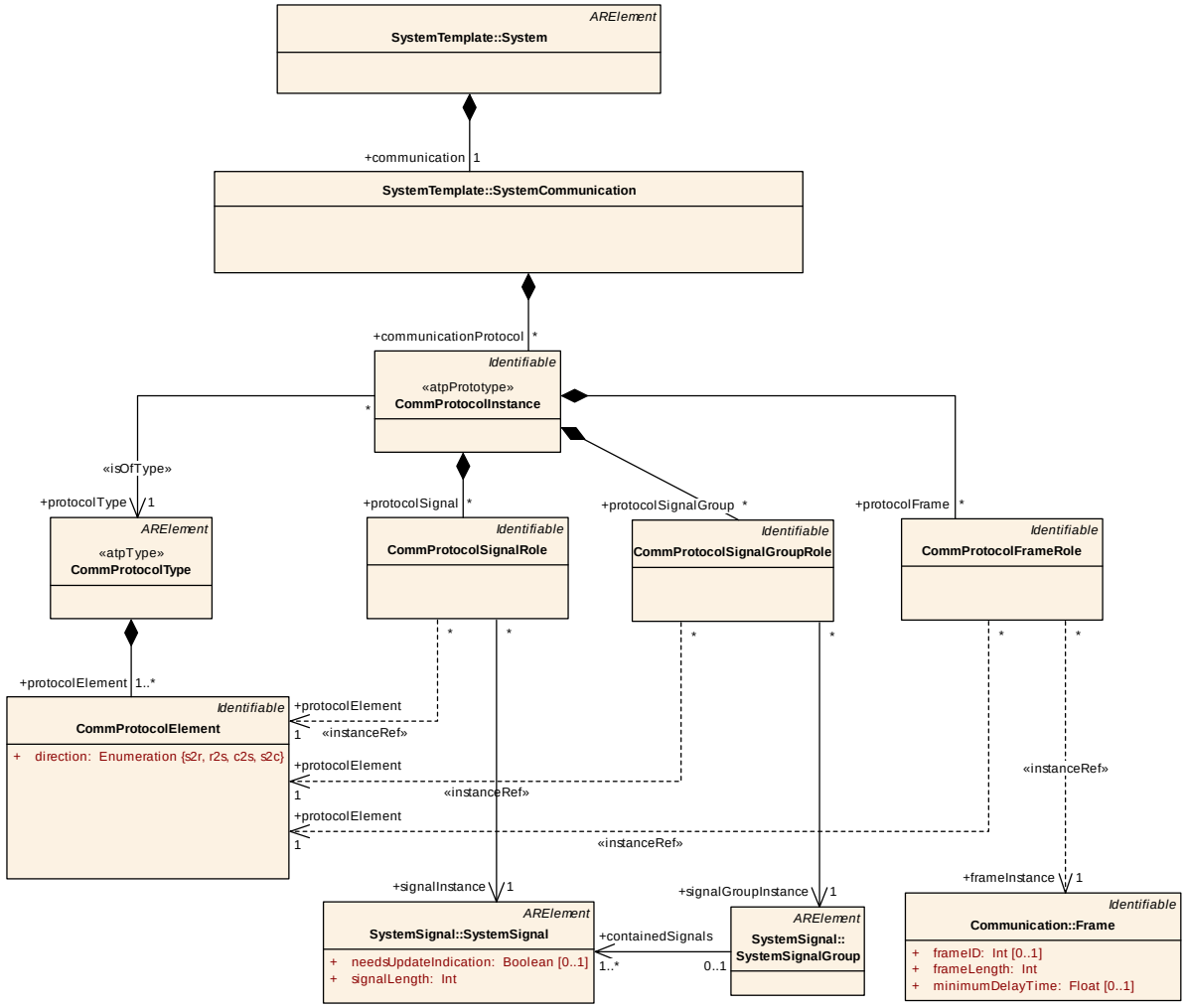

Figure 26: Communication Protocols (CommunicationProtocols)

The protocol type definition consists of the following elements:

Class	CommProtocolType			
Package	M2::AUTOSARTemplates::SystemTemplate::CommunicationProtocol			
Class Description	The protocol type description describes all necessary protocol parameters that are the same for all instances of the protocol and that are necessary to configure the protocol., This is e.g.. a generic definition of ISO 15765-2.			
Base Class(es)	ARObject, Identifiable, PackageableElement, NonSplitableElement, ARElement			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
protocolElement	CommProtocolElement	1..*	aggregation	The communication protocol type description includes at least one protocol element.

Class	CommProtocolElement			
Package	M2::AUTOSARTemplates::SystemTemplate::CommunicationProtocol			
Class Description	Description of the protocol elements. These may be frames, signals (which are in turn part of a frame) or signal groups.			
Base Class(es)	ARObject, Identifiable			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
direction	CommProtocolElementDirection-Enum	1	aggregation	The direction in which the element is sent- s2r : Sender To Receiver (For Sender-Receiver Communication, e.g. data elements) r2s: Receiver To Sender (For Sender-Receiver Communication, e.g. Acknowledgements) c2s: Client to Server (For Client-Server Communication, "in" parameters) s2c Server to Client (For Client-Server Communication, "out" parameters)

The communication protocol instance definition is defined as follows:

Class	CommProtocolInstance			
Package	M2::AUTOSARTemplates::SystemTemplate::CommunicationProtocol			
Class Description	An instance of a communication protocol, used for exchanging information between specific communication partners. This might e.g. be an instance of ISO 15765-2 running between ECU A and ECU B.			
Base Class(es)	ARObject, Identifiable			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
protocolFrame	CommProtocolFrameRole	0..*	aggregation	The communication protocol instance can uses frames as protocol elements.
protocolSignal	CommProtocolSignalRole	0..*	aggregation	The communication protocol instance can uses signals as protocol elements.
protocolSignalGroup	CommProtocolSignalGroupRole	0..*	aggregation	The communication protocol instance can uses signal groups as protocol elements.
protocolType	CommProtocolType	1	reference to type	reference to the communication protocol that is used in this instance.

Class	CommProtocolSignalRole
Package	M2::AUTOSARTemplates::SystemTemplate::CommunicationProtocol
Class Description	If the protocol instance uses signal instances (in a specific frame instance) as protocol elements, they are specified with this element.
Base Class(es)	ARObject, Identifiable

Attribute	Datatype	Mul.	Link Type	Attribute Description
protocolElement	CommProtocolSignalRole_protocol-Element	1	reference to instance	Reference to the protocol element in the protocol type that is implemented by this signal role.
signalInstance	SystemSignal	1	reference	Reference to the system signal used to transport the communication protocol. Each signal Role Description refers to exactly one system signal, but the same system signal may be included in different role descriptions. The latter is the case, if the same system signal is e.g. used by different protocols (that then will have additional means to indicate which protocol is active)

Class	CommProtocolFrameRole			
Package	M2::AUTOSARTemplates::SystemTemplate::CommunicationProtocol			
Class Description	If the protocol instance uses frame instances as protocol elements, they are specified with this element. This is e.g. the case in protocol ISO 15765-2, where complete frames are used to exchange information between the communication partners.			
Base Class(es)	ARObject, Identifiable			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
frameInstance	CommProtocolFrameRole_frame-Instance	1	reference to instance	Reference to the frame instance. Each frame role description refers to exactly one frame instance, but the same signal instance may be included in different role descriptions. The latter is the case, if the same frame instance is e.g. used by different protocols (that then will have additional means to indicate which protocol is active)
protocolElement	CommProtocolFrameRole_protocol-Element	1	reference to instance	Reference to the protocol element in the protocol type that is instantiated by this frame role.

Class	CommProtocolSignalGroupRole			
Package	M2::AUTOSARTemplates::SystemTemplate::CommunicationProtocol			
Class Description	If the protocol instance uses signal groups as protocol elements, they are specified with this element.			
Base Class(es)	ARObject, Identifiable			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
protocolElement	CommProtocolSignalGroupRole_-protocolElement	1	reference to instance	Reference to the protocol element in the protocol type that is instantiated by this signalGroup role.
signalGroupInstance	SystemSignalGroup	1	reference	Reference to the signalGroup. Each signalGroup role description refers to exactly one signalGroup, but the same signalGroup may be included in different role descriptions. The latter is the case, if the same signalGroup is e.g. used by different protocols (that then will have additional means to indicate which protocol is active)

5.6.3 Examples continued: communication protocol definitions using the Meta model

In the following, the ISO15765 and simple protocols are used to illustrate the usage of the communication protocol Meta model.
The first figure shows an example usage of ISO15765:

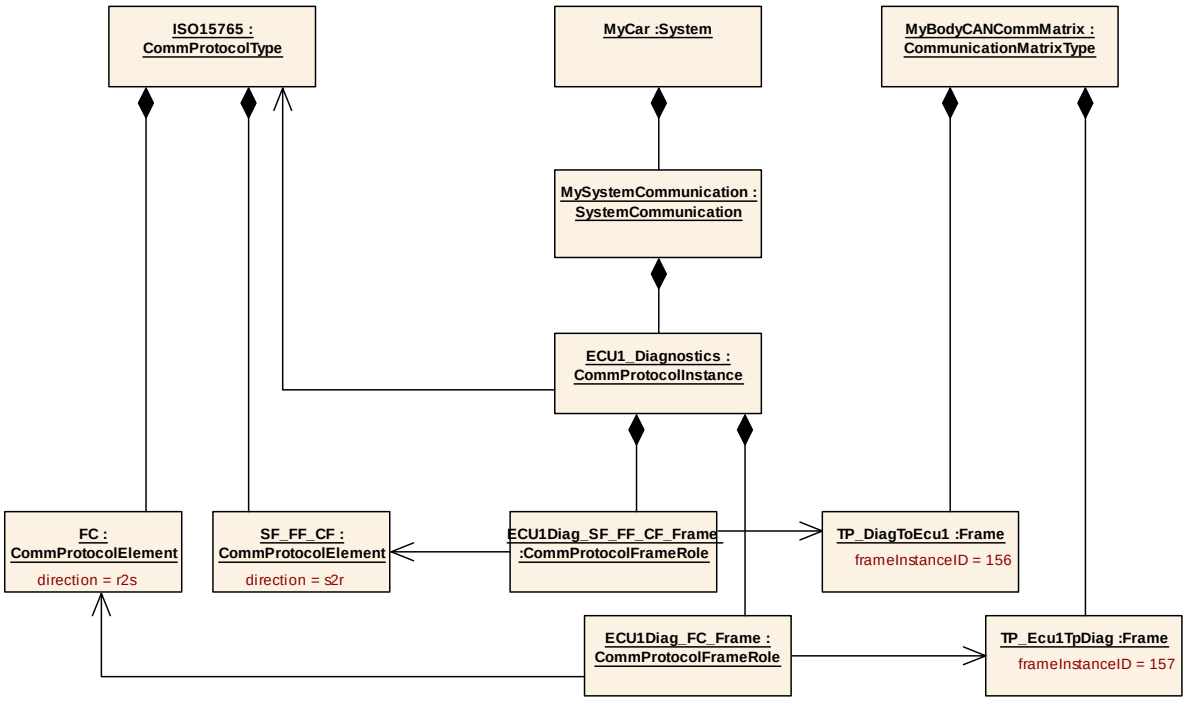

Figure 27: Example definition and usage of ISO15765 transport protocol (CommProtocolExampleISO)

To the left, the protocol type is defined. Two protocol elements are defined: SF_FF_CF to hold single frame, first frames and consecutive frames sent from sender to receiver, and FC used for flow control from receiver to sender.
To the right, an extract of the communication matrix of the system is shown with two frames that will be used to transport ISO15765 messages to and from ECU1.
Finally in the center, the communication protocol instance is depicted. It defines two protocol frame roles, again one for the SF, FF, and CF messages which references to SF_FF_CF of the communication protocol type to indicate that this frame will be used to transport ISO15765 SF_FF_CF elements, and references to frame TP_DiagToEcu1 to indicate that this frame will be used to transport the information.
For a standardized protocol like ISO15765, the communication protocol type will be well known to the different tools that generate ECU code. Thus, the protocol instance defined carries enough information so that these tools can derive what data must be mapped where and can work out the details of the mapping.
The second figure shows an example for the simple ACK protocol described above.

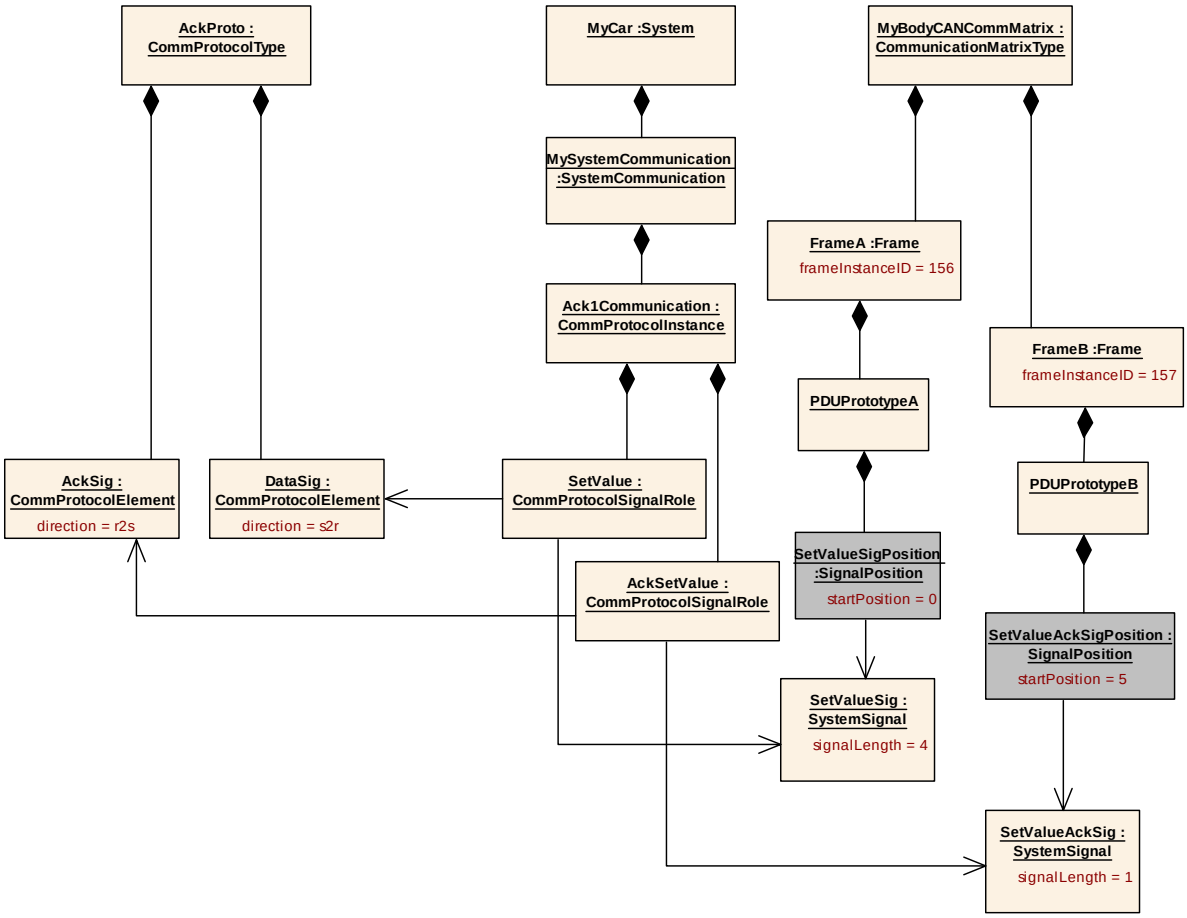

Figure 28: Example definition and usage of the ACK protocol (CommProtocolExampleAckProtocol)

This time, the protocol does not use complete frames, but only individual system signals within a PDU. The communication protocol type does not change and looks similar as in the previous example. But in the communication protocol instance, `CommProtocolSignalRoles` are used instead of the frame roles, since we need to refer to individual signals. To the right, the frames are defined again with the system signals implicitly referenced from the PDUs inside these frames.[3]

[3] The signal positions are shown in grey since they are not really aggregated by PDUPrototypes, but by the PDUTypes.

6 Mapping

A central part of the system generation process is the mapping of software components to ECUs, and the subsequent mapping of the communication between these SW components to bus frames. Input to the SW component mapping are the software composition, which describes which SW components have to be mapped, and the System Topology, which defines the ECU instances that are available as mapping targets. Once this mapping is done, also the communication matrix has to be taken into account for the next mapping step, the mapping of data elements exchanged between SW components to bus frames. This communication matrix may either be predefined, or may be generated as part of this second mapping step.
In the metamodel, different aspects of these mapping are aggregated by the meta class `System Mapping`, as shown in **Figure 29**.

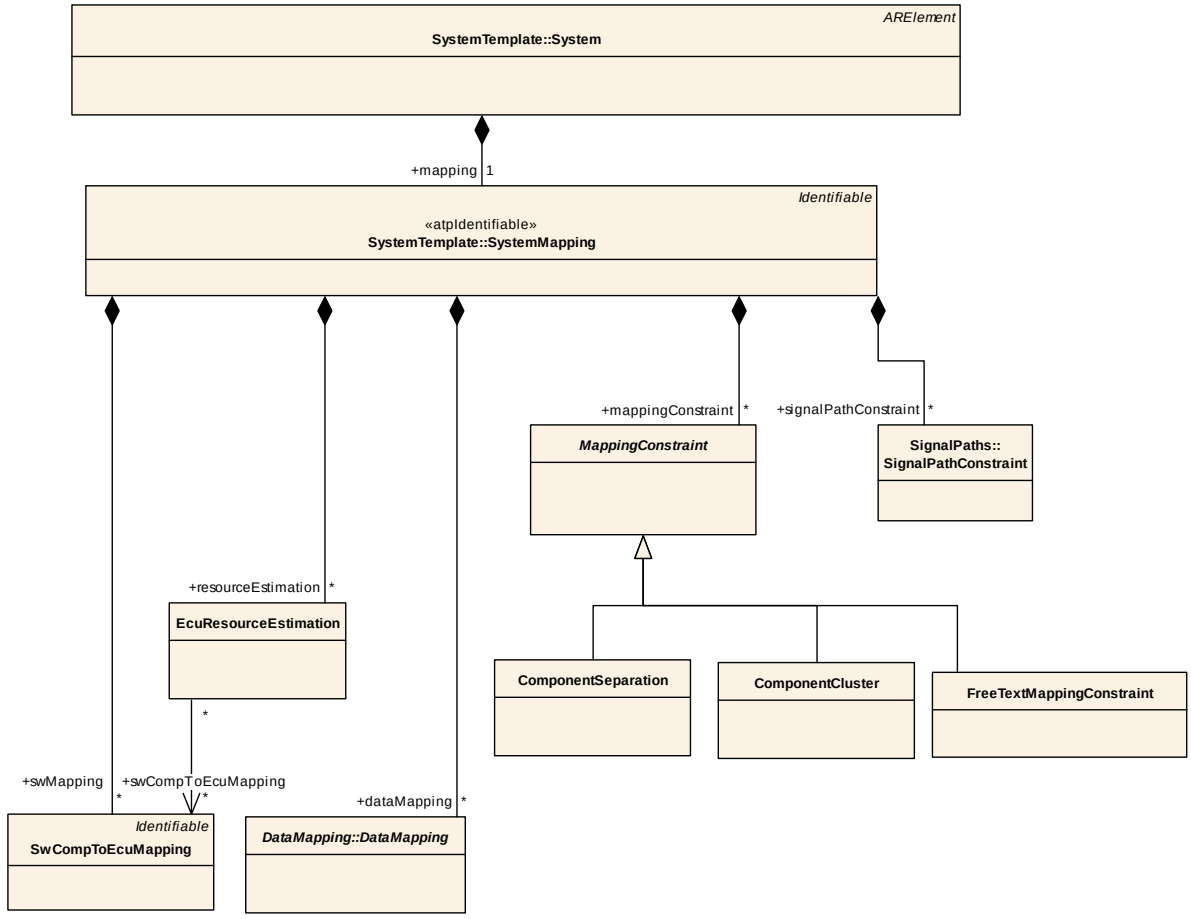

Figure 29: Mapping Overview (Mapping)

MetaClass `SWCompToEcuMapping` is used to define mappings of SW components to ECUs (see chapter 6.1). Meta class `DataMapping` is used to define mappings of data to communication using a communication matrix (see chapter 6.2) and the resource estimation for RTE and basic software is specified using class EcuResourceEstimation (see chapter 6.3). Meta class `MappingConstraint` is used to define constraints that constrain the mapping of SW components, see chapter

6.1.3. Finally, meta class `SignalPathConstraint` is used to define constraints that constrain the data mapping, see chapter 6.2.2.

Class	**SystemMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Class Description	The system mapping aggregates all mapping aspects (mapping of SW components to ECUs, mapping of data elements to signals, and mapping constraints).			
Base Class(es)	ARObject, Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
dataMapping	DataMapping	0..*	aggregation	The data mappings defined.
mappingConstraint	MappingConstraint	0..*	aggregation	Constraints that limit the mapping freedom for the mapping of SW components to ECUs.
resourceEstimation	EcuResourceEstimation	0..*	aggregation	Resource estimations for this set of mappings, zero or one per ECU instance.
signalPathConstraint	SignalPathConstraint	0..*	aggregation	Constraints that limit the mapping freedom for the mapping of data elements to signals.
swImplMapping	SwCompToImplMapping	0..*	aggregation	The mappings of AtomicSoftwareComponent Instances to Implementations.
swMapping	SwCompToEcuMapping	0..*	aggregation	The mappings of SW components to ECUs.

6.1 Software Component Mapping

A fundamental concept of AUTOSAR is that SW components may be developed independently of a specific ECU hardware, and can be mapped to an ECU in the AUTOSAR System Generation Process. The System Constraint Description acts as an input to this System Generation Phase. Nevertheless, there may be some SW components which are already mapped due to previous iterations of the system generation step, and there may be system constraints that limit the system architect's freedom to map SW components to arbitrary ECUs. In the following, the individual elements are described in more detail.

6.1.1 SW Component to ECU Mapping

With `SwCompToEcuMapping` element it is possible to express the mapping of Software components to one ECU instance. **Figure 30** shows this structure.

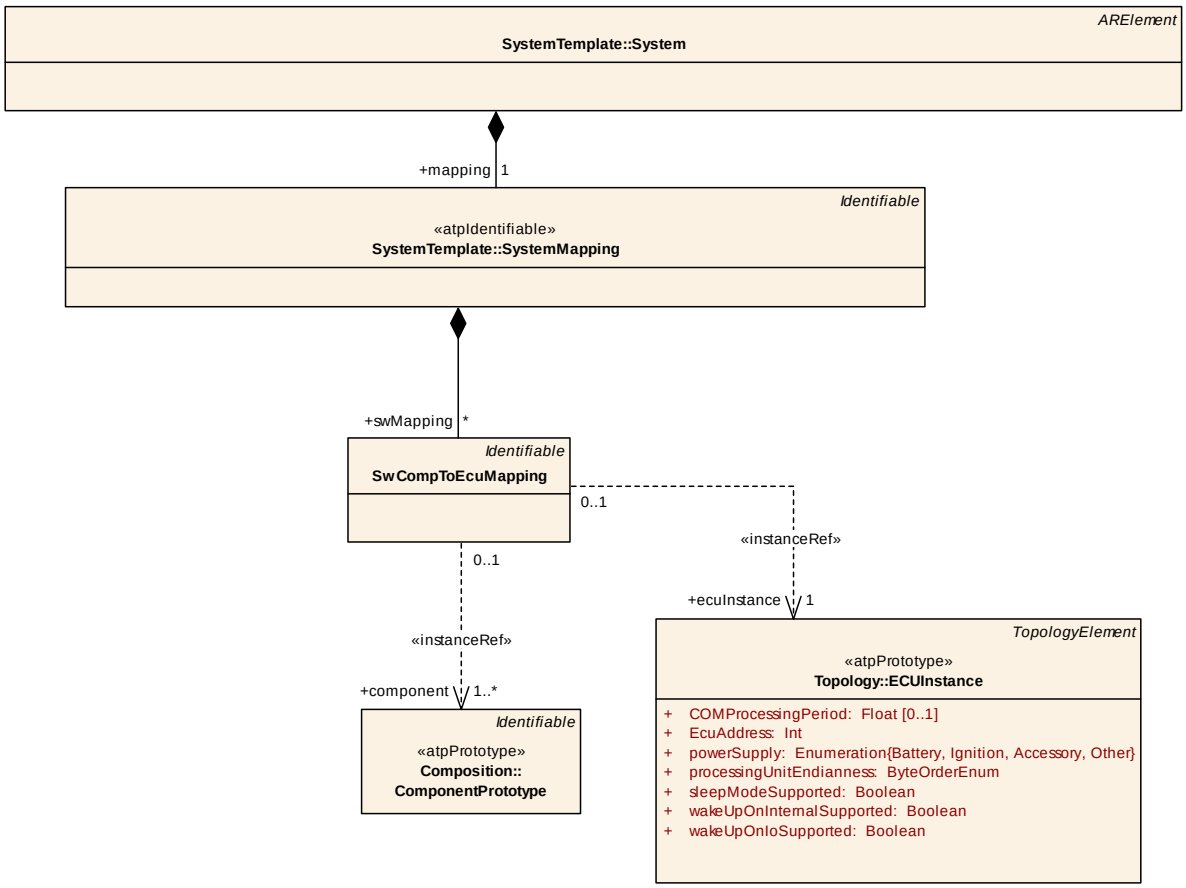

Figure 30: SW component to ECU mapping (SwCompToEcuMapping)

The following table describes the `SwCompToEcuMapping` in detail.

Class	**SwCompToEcuMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::SWmapping			
Class Description	Map software components to a specific ECU			
Base Class(es)	ARObject, Identifiable			
Attribute	*Datatype*	*Mul.*	*Link Type*	*Attribute Description*
component	SwCompToEcuMapping_component	1..*	reference to instance	References to the software component instances that are mapped to the referenced ECUInstance. If the component prototype referenced is a composition, this indicates that all atomic software components within the composition are mapped to the ECU.
ecuInstance	SwCompToEcuMapping_ecuInstance	1	reference to instance	EcuInstance is a reference to an ECU Instance description

Usage of Template to describe the "System Constraints"
The mapping of SW Components to ECUs can be predefined. The predefinition will force the system generator to use the specified mapping. Thus, with the `SwCompToEcuMapping` element it is possible to describe that one or more SW Components must be mapped to a specific ECU.

Usage of Template to describe the "System Configuration"
In a completed System Configuration Description, all SW components are mapped to ECUs. The mapping in the System Configuration Description is described by one `SwCompToEcuMapping` element for each `ECUInstance` used in the system.

6.1.2 Software Component to Implementation Mapping

As several implementations may exist for the same `AtomicSoftwareComponentType`, it needs to be decided on and specified which instances of a given `AtomicSoftwareComponentType` are mapped to which `Implementation`. According to the AUTOSAR Methodology this information can either be added within the `Configure System` activity, or later when the RTE part is configured during `Configure ECU` phase. If the mapping is done in System Configuration, a `SwCompToImplMapping` is being used for assigning one `Implementation` to one or more instances of `ComponentPrototype` relating to the same `AtomicSoftwareComponentType`. This is illustrated in **Figure 31** below.

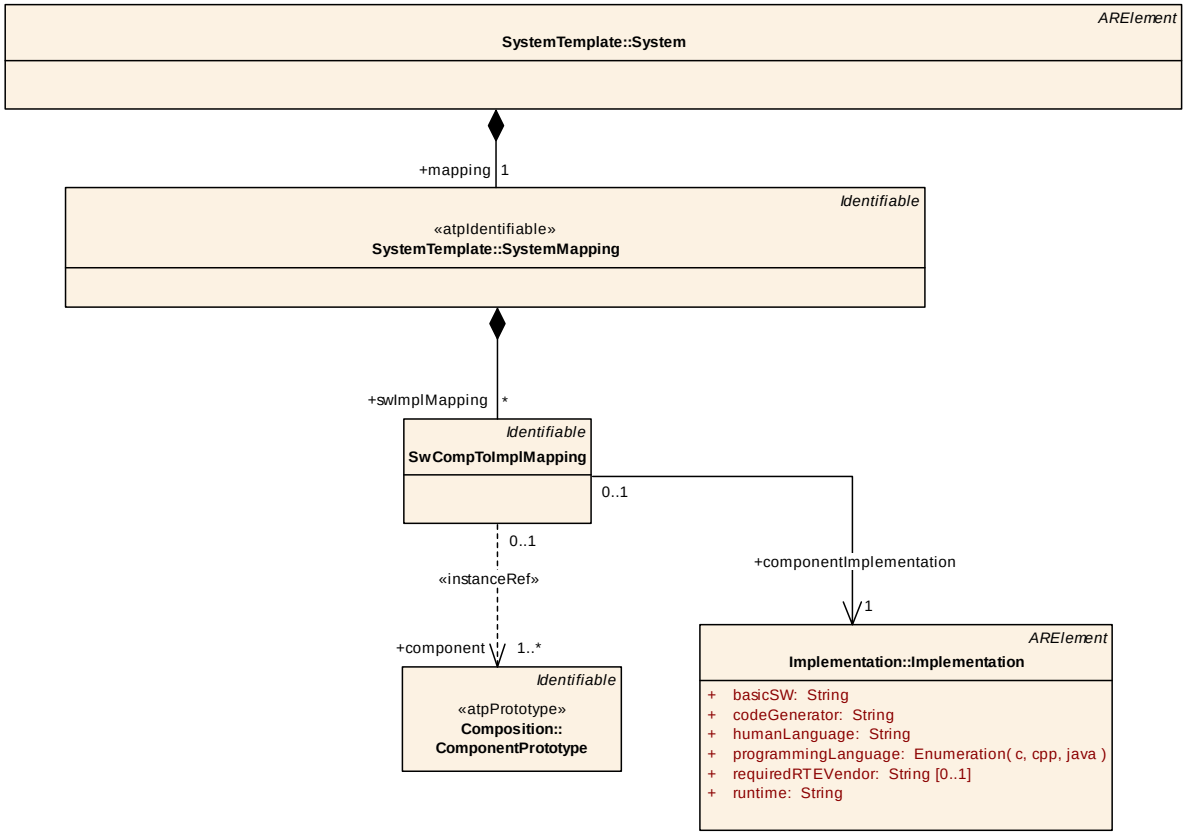

Figure 31: SW Component to Implementation mapping (SwCompToImplMapping)

The following table contains the detailed description of SwCompToImplMapping:

Class	**SwCompToImplMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::SWmapping			
Class Description	Map instances of an AtomicSoftwareComponentType to a specific Implementation.			
Base Class(es)	ARObject, Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
componentImplementation	Implementation	1	reference	Implementation to be used by the specified SW component instance. This allows to achieve more precise estimates for the resource consumption that results from mapping the instance of an atomic SW component onto an ECU.
component	SwCompToImplMapping_component	1..*	reference to instance	Reference to the software component instances that are being mapped to the specified Implementation. The targeted ComponentPrototype needs be of the AtomicSoftwareComponentType being implemented by the referenced Implementation.

6.1.3 Software Component Mapping Constraints

In contrast to the mapping description described in the previous chapter, mapping constraints allow to define invariants that have to be fulfilled by a valid mapping. They are aggregated in the `MappingConstraint` element as introduced in chapter 3.5 and depicted **Figure 11**.

Usage of Template to describe the “System Constraints”
There may be system constraints that limit the system generators freedom to map SW components to arbitrary ECUs. This system constraints can be necessary e.g. for optimization and safety reasons to make additional guidelines for the System Generator. This chapter describes which mapping constraints can be described in the System Constraint Description.

Usage of Template to describe the “System Configuration”
After the mapping has been completed, the system configuration will contain mapping descriptions for all elements, and the mapping constraints are obsolete. But that does not mean that mapping constraints have to be deleted after the system generation step. By deleting the mapping constraints you would lose the information why a mapping of a SW Component to an ECU is chosen.
The detailed description of this meta class can be found in the following table:

Class	**MappingConstraint {abstract}**
Package	M2::AUTOSARTemplates::SystemTemplate::SWmapping
Class Description	Different constraints that may be used to limit the mapping of SW components to ECUs.
Base Class(es)	ARObject

6.1.3.1 ComponentCluster and ComponentSeperation

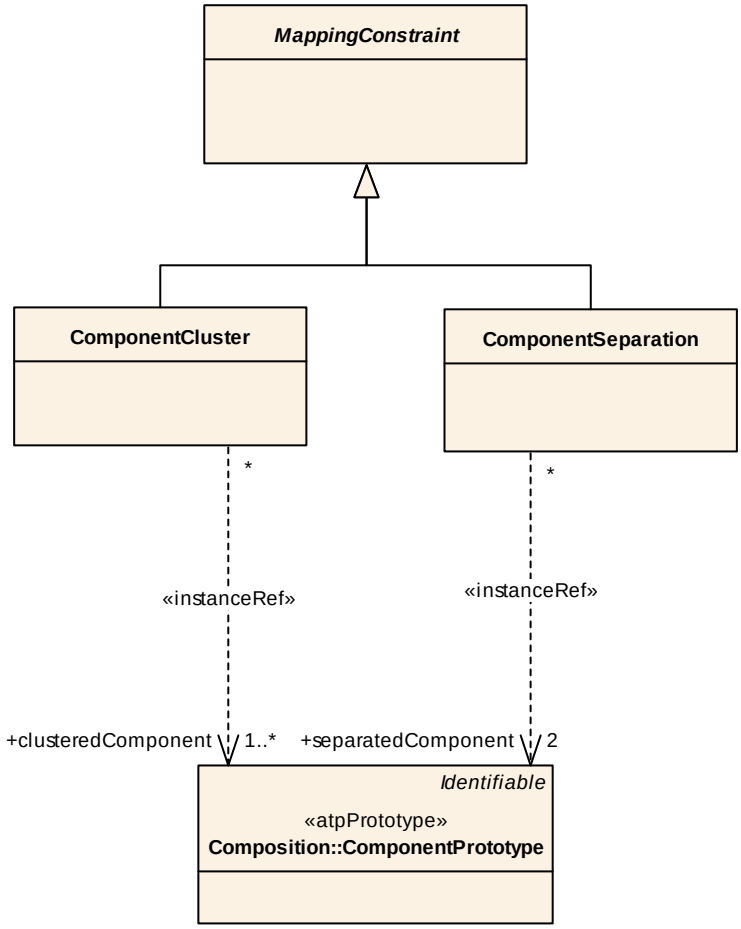

Figure 32: Details on ComponentCluster and ComponentSeparation (SwCompClustering)

These two constraints express the restrictions that SW components impose each other when performing the mapping onto the ECUs. In fact, before the mapping process begins, it can be useful to impose the allocation of a predefined set of SW components onto the same ECU, especially if such a set is tightly linked from a functional point of view. In the same way, two critical SW components, performing some kind of redundancy, may be not suitable to run both on the same ECU. Thus, we call these two kinds of mapping constraints, respectively, `ComponentCluster` and `ComponentSeparation`.

6.1.3.1.1 ComponentCluster

The `ComponentCluster` constraint (also, *clustering*) is to be used for expressing that a certain set of SW components (atomic or not) must be mapped (allocated) onto the same ECU. This is some kind of “execute together on same ECU” constraint.
The semantic of the clustering constraint is straightforward if all concerned SW components are atomic. Otherwise, it shall be interpreted as follows: all of the atomic SW components making up the composition must be mapped together onto the same ECU together with all other SW components (atomic or not) affected by the constraint. This also means that a *clustering* constraint can also refer to only a single composition.
A *clustering* constraint is part of a `MappingConstraint` element and it must refer to one or more `ComponentPrototype` elements, representing the instances of the SW component(s) that must be mapped together.

Class	**ComponentCluster**			
Package	M2::AUTOSARTemplates::SystemTemplate::SWmapping			
Class Description	Constraint that forces the mapping of all referenced SW component instances to the same ECU			
Base Class(es)	ARObject, MappingConstraint			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
clusteredComponent	ComponentCluster_clustered-Component	1..*	reference to instance	Reference to the components that have to be mapped together.

6.1.3.1.2 ComponentSeparation

The `ComponentSeparation` constraint (also, *separation*) is to be used for expressing that two SW components (atomic or not) shall not be mapped (allocated) onto the same ECU. This is some kind of “do not execute together on same ECU” constraint.
The semantic of the separation constraint is straightforward if one or both SW components are atomic. Otherwise, it shall be interpreted as follows: any of the atomic SW components making up the first composition, must not be mapped onto the same ECU with any atomic SW component from the second composition. As a consequence, and to preserve consistency, an atomic SW component instance cannot be part of two compositions concerned by the same separation constraint, i.e. the two compositions have to be disjoint with regards to component instances[4].
A *separation* constraint is part of a `MappingConstraint` element and it must refer to two `ComponentPrototype` elements, representing the two SW component instances that must not be allocated together.

[4] The only case where a component instance could be in both sets is if the ComponentSeparation refers to two elements where one of them is a substructure of the other. Consider the case that Atomic SW Component A is aggregated by composition B, which in turn is aggregated by composition C. Then instance A is both in B and C. It is not a good idea to formulate a separation constraint stating that B and C should not be on the same ECU.

Class	**ComponentSeparation**			
Package	M2::AUTOSARTemplates::SystemTemplate::SWmapping			
Class Description	Constraint that forces the two referenced SW components (called A and B in the following) not to be mapped to the same ECU. If a SW component (e.g. A) is a composition, none of the atomic SW components making up the A composition must be mapped together with any of the atomic SW components making up the B composition. Furthermore, A and B must be disjoint.			
Base Class(es)	ARObject, MappingConstraint			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
separatedComponent	ComponentSeparation_separated-Component	2	reference to instance	The two components that have to be mapped to different ECUs

6.2 Data Mapping

The data mapping description may either be mapping of client server communication or sender receiver communication. The different mappings are described in the following chapters in detail.

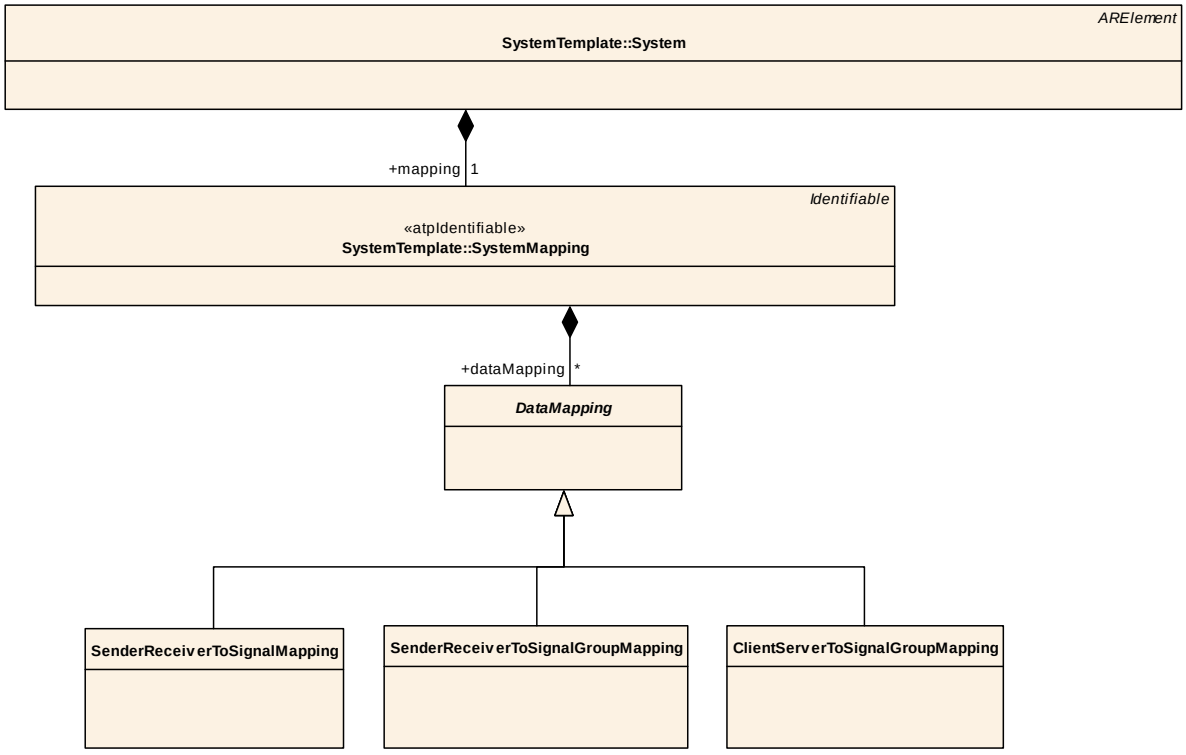

Figure 33: Overview: Data Mapping Description (DataMappingOverview)

Usage of Template to describe the “System Constraints”
The System Constraint Description may describe the predefined mapping of SW Components to certain ECUs (see chapter 6.1.1 above). Only if such a mapping

exists, it is also reasonable to define the mapping of the data exchanged between those mapped SW components by a predefined mapping of data elements to the Communication Matrix.
Usage of Template to describe the “System Configuration”
In contrary to the System Constraint Description the final System Configuration Description contains all data mapping definitions. No matter if they were predefined (system constraint) or if they were generated by the System-Generator.

6.2.1 Mapping of DataPrototypes on SystemSignals

The `SystemSignal` represents the communication system's view of data exchanged between SW components which reside on different ECUs. The following chapter describes how DataPrototypes, being the units of information to be transported between providing and requiring ports, are mapped onto SystemSignals.
In the Software part of the System Template a top-level `SoftwareComposition` is expressed by using `AssemblyConnectorPrototypes` and `Delegation-ConnectorPrototypes` to connect the `PPortPrototypes` and `RPortPrototypes` of `ComponentPrototypes` with each other on the VFB-level.
Ultimately, each chain of `ConnectorPrototypes` leads to exactly one `PPortPrototype`. This `PPortPrototype` references a `PortInterface`, which may either be a `SenderReceiverInterface` or a `ClientServerInterface`. It is the task of SystemConfiguration to map each `DataElement` or `Argument-Prototype` contained in these Ports referenced by the `ConnectorPrototype` onto a `SystemSignal`. However, the same `SystemSignal` may satisfy more than one connector, and one connector may be implemented by several `SystemSignals` (e.g. one per `DataElement` in the `PortInterface` being connected), so there is no 1:1 mapping between `AssemblyConnectors` and `SystemSignals`. Therefore, if one needs to find all `SystemSignals` implementing a particular `Assembly-Connector`, this requires a model query which compares the `ProvidedPort` end of the connector chain with the `PortPrototype` providing the `DataElement`
In the following sections, each reference to a `DataElementPrototype` or `ArgumentPrototype` is of type Instance Reference[5]. This means it not only references the actual `DataElementPrototype`, but additionally contains contextual references to the `PortPrototype` and the hierarchy of `ComponentPrototypes` forming the individual instance context of the `DataElementPrototype`. Therefore the abovementioned query requires a comparison of the full instance reference paths of the connector end and the PortPrototype context of the DataElement to be mapped to the signal.

6.2.1.1 Mapping of Data Elements with primitive datatypes on SystemSignals (Sender-Receiver Communication)

The `SystemSignal` represents the communication system's view of data exchanged between SW components which reside on different ECUs. The `DataElementPrototype` meta-class is defined in the SW Component Template. The datatype of the data element may be a primitive one or a composite one.

Primitive data types cannot be decomposed in other data types. The composite data types "array" and "record" provide the means to build new data types.
This chapter describes the relation between the `DataElementPrototypes` with primitive datatypes and the `SystemSignal`.
In a complete System Template, it is sufficient to refer to the data element in the ProvidePort to define the mapping of the communication between a provider and its receivers. This is possible since the connectors implicitly define which RequirePorts are connected to the ProvidePort. In an ECU extract of the system description, where only the relevant parts of the SW compositions are defined, it is in some cases also necessary to refer to RequirePorts, if the corresponding ProvidePort is not part of the extract. This is described in more detail in chapter 7.2.

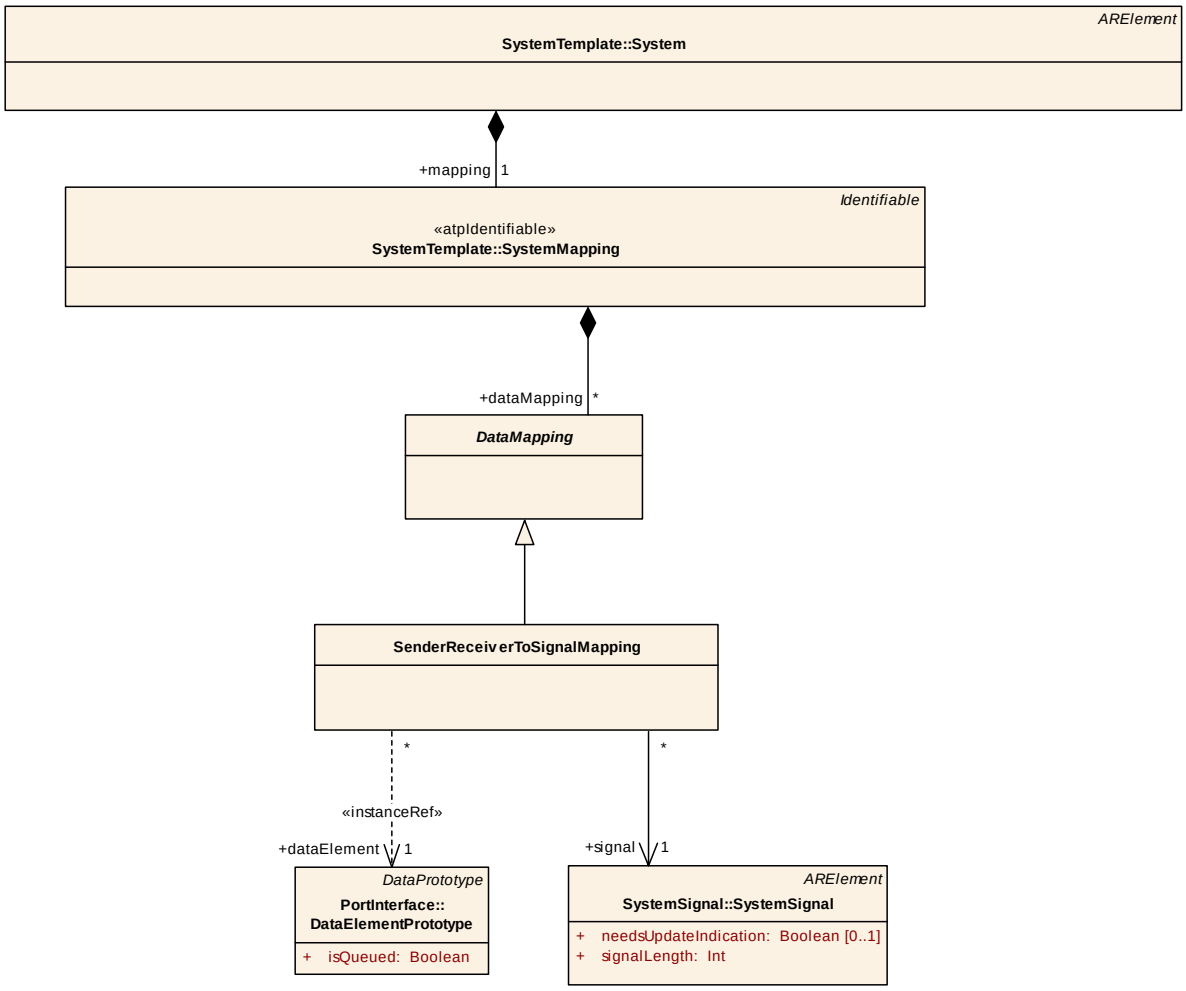

Figure 34: Mapping of data elements with primitive datatypes (SenderRecPrimitiveTypeMapping)

Class	**SenderReceiverToSignalMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Class Description	Mapping of a sender receiver communication data element with a primitive datatype to a signal. If the data element has to be transmitted in several signals (e.g. if the data is sent out to different receivers in different signals), one have to define several mappings.			
Base Class(es)	ARObject, DataMapping			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
dataElement	SenderReceiverToSignalMapping_-dataElement	1	reference to instance	Reference to the data element, which ought to be sent over the Communication bus. This DataElement is described in the Software Component Template.
signal	SystemSignal	1	reference	Reference to the system signal used to carry the data element.

6.2.1.2 Mapping of Data Elements with composite datatypes on SignalGroups (Sender-Receiver Communication)

This chapter describes the mapping of `DataElementPrototypes` with composite datatypes to `SystemSignals`.
The RTE is required to treat AUTOSAR signals transmitted using sender-receiver communication atomically. To achieve this, the “signal group” mechanisms shall be utilized. The complex data type must be decomposed into single signals. As this set of single signals has to be treated as atomic, it is placed in a “signal group”.
Thus, each `PrimitiveType` will be one `SystemSignal` in the System Description. For a CompositeType several `SystemSignals` will be used. The relationship between the `SystemSignals` and the `DataElementPrototypes` is provided in the `SenderReceiverToSignalGroupMapping`.

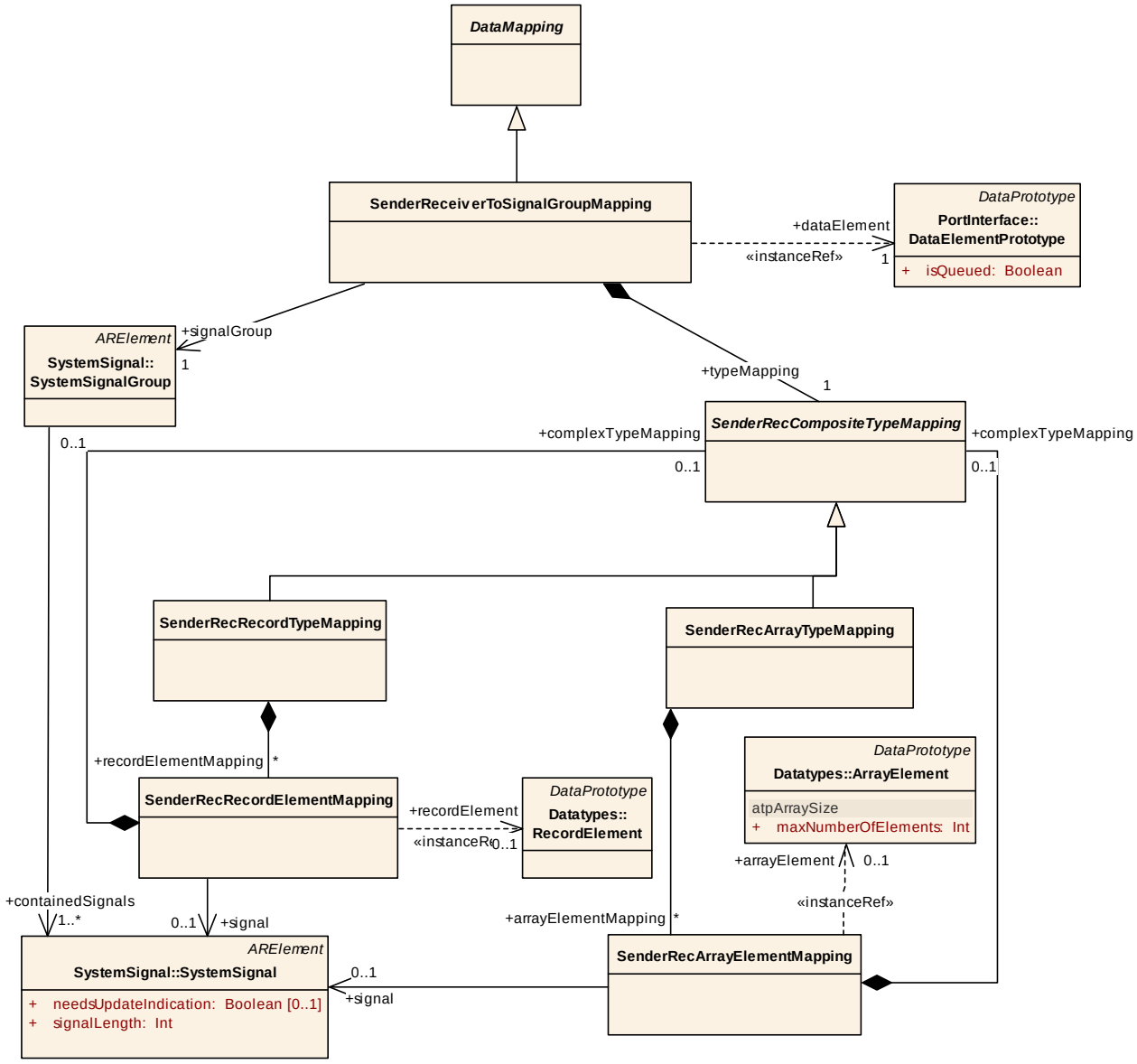

Figure 35: Mapping of data elements with composite datatypes (SenderRecCompositeTypeMapping)

Class	**SenderReceiverToSignalGroupMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Class Description	Mapping of a sender receiver communication data element with a composite datatype to a signal group.			
Base Class(es)	ARObject, DataMapping			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
dataElement	SenderReceiverToSignalGroup-Mapping_dataElement	1	reference to instance	Reference to the data element, which ought to be sent over the Communication bus.
signalGroup	SystemSignalGroup	1	reference	Reference to the signal group, which contain all primitive datatypes of the composite type
typeMapping	SenderRecCompositeTypeMapping	1	aggregation	

Class	**SenderRecCompositeTypeMapping {abstract}**
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping
Class Description	Two mappings exist for the composite data types: "ArrayTypeMapping" and "RecordTypeMapping". In both, a primitive datatype will be mapped to a system signal. But it is also possible to combine the arrays and the records, so that an "array" could be an element of a "record" and in the same manner a "record" could be an element of an "array". Nesting these data types is also possible. If an element of a composite data type is again a composite one, the "CompositeTypeMapping" element will be used one more time (aggregation between the ArrayElementMapping and CompositeTypeMapping or aggregation between the RecordElementMapping and CompositeTypeMapping).
Base Class(es)	ARObject

Class	**SenderRecArrayTypeMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Class Description	If the compositeType is an Array, the "ArrayTypeMapping" will be used.			
Base Class(es)	ARObject, SenderRecCompositeTypeMapping			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
arrayElementMapping	SenderRecArrayElementMapping	0..*	aggregation	Each ArrayElement must be mapped on a SystemSignal.

Class	**SenderRecRecordTypeMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Class Description	If the compositeType is a Record, the "RecordTypeMapping" will be used.			
Base Class(es)	ARObject, SenderRecCompositeTypeMapping			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
recordElementMapping	SenderRecRecordElementMapping	0..*	aggregation	Each RecordElement must be mapped on a SystemSignal.

Class	**SenderRecRecordElementMapping**
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping
Class Description	Mapping of a primitive record element to a SystemSignal. If the element is composite, there will be no mapping (multiplicity 0). In this case the "RecordElementMapping" Element will aggregate the "TypeMapping" Element. In that way also the composite datatypes can be mapped to SystemSignals.
Base Class(es)	ARObject

Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
complexTypeMapping	SenderRecCompositeTypeMapping	0..1	aggregation	This aggregation will be used if the element is composite.
recordElement	SenderRecRecordElementMapping_-recordElement	0..1	reference to instance	Reference to an element in a record.
signal	SystemSignal	0..1	reference	Reference to the system signal used to carry the primitive RecordElement.

Class	**SenderRecArrayElementMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Class Description	The ArrayElement may be a primitive one or a composite one. If the element is primitive, it will be mapped to the "SystemSignal" (multiplicity 1). If the element is composite, there will be no mapping to the "SystemSignal" (multiplicity 0). In this case the "ArrayElementMapping" Element will aggregate the "TypeMapping" Element. In that way also the composite datatypes can be mapped to SystemSignals.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
arrayElement	SenderRecArrayElementMapping_-arrayElement	0..1	reference to instance	Reference to an element in an array.
complexTypeMapping	SenderRecCompositeTypeMapping	0..1	aggregation	This aggregation will be used if the element is composite.
signal	SystemSignal	0..1	reference	Reference to the system signal used to carry the primitive ArrayElement.

6.2.1.3 Mapping of Client Server Operations to Signal Groups

The Client/Server interfaces aggregate a number of operations. Each description of an operation consists of the description of its arguments. Furthermore, the RTE is responsible to map a response to the corresponding request. For this mapping transaction handles are used. The transaction handle contain a client identifier and a sequence counter.
The arguments, application errors, client identifier and sequence counter of an operation are mapped to `SystemSignals` of two dedicated `SystemSignalGroup` elements; one for the request and one for the response. The RTE Client Server Protocol is used to provide a specific semantics to each of these `SystemSignalGroups` and `SystemSignals`, also those which are introduced only to support the protocol. This is described in more detail in [11].
The datatype of an argument may be a primitive one or a composite one. Each primitive argument will be mapped directly onto one `SystemSignal`. The complex data type must be decomposed into single signals.
The relationship between the `SystemSignals` and the `Arguments` is provided in the `ClientServerToSignalGroupMapping`.
In a complete System Template, it is sufficient to refer to the operation in the ProvidePort to define the mapping of the communication between a provider and its receivers. This is possible since the connectors implicitly define which RequirePorts are connected to the ProvidePort. In an ECU extract of the system description, where only the relevant parts of the SW compositions are defined, it is in some cases also necessary to refer to RequirePorts, if the corresponding ProvidePort is not part of the extract. This is described in more detail in chapter 7.2.

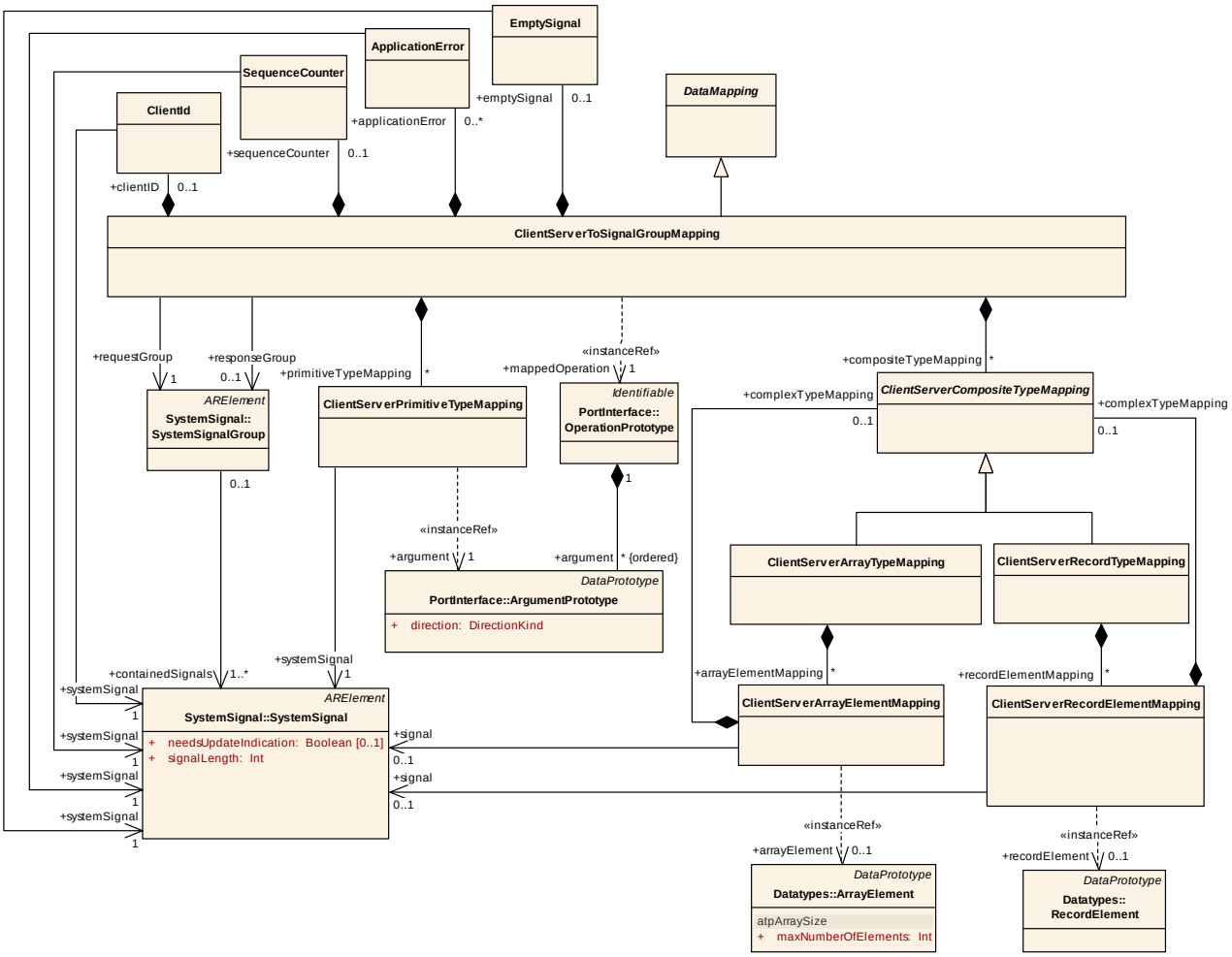

Figure 36: Operation Mapping (ClientServerOperationMapping)

Class	**ClientServerToSignalGroupMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Class Description	Mapping of client server operation arguments to signals of a signal group. Arguments with a primitive datatype will be mapped via the "ClientServerPrimitiveTypeMapping" element. Arguments with composite datatypes will be mapped via the "CompositeTypeMapping" element.			
Base Class(es)	ARObject, DataMapping			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
applicationError	ApplicationError	0..*	aggregation	
clientID	ClientId	0..1	aggregation	
compositeTypeMapping	ClientServerCompositeType-Mapping	0..*	aggregation	
emptySignal	EmptySignal	0..1	aggregation	
mappedOperation	ClientServerToSignalGroup-Mapping_mappedOperation	1	reference to instance	Reference to the operation whose arguments should be transmitted via the communication bus.
primitiveTypeMapping	ClientServerPrimitiveType-Mapping	0..*	aggregation	Mapping of an argument with a primitive datatype to a signal.
requestGroup	SystemSignalGroup	1	reference	Reference to the signal group which contains the references to request signals used to transport the OUT arguments of the operation or the empty signal if the operation doesn't have OUT arguments.
responseGroup	SystemSignalGroup	0..1	reference	Reference to the signal group which contains the references to response signals used to transport the IN arguments of the operation.
sequenceCounter	SequenceCounter	0..1	aggregation	

Class	ClientId			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Class Description	In case of a server on one ECU with multiple clients on other ECUs, the client server communication shall use different unique COM signals and signal groups for each client to allow the identification of the client associated with each system signal.			
Base Class(es)	ARObject			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
systemSignal	SystemSignal	1	reference	Reference to the SystemSignal with the ClientID.

Class	SequenceCounter			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Class Description	The purpose of sequence counters is to map a response to the correct request of a known client.			
Base Class(es)	ARObject			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
systemSignal	SystemSignal	1	reference	Reference to the SystemSignal with the SequenceCounter.

Class	ApplicationError			
Package	M2::AUTOSARTemplates::SWComponentTemplate::VFBErrors			
Class Description	This is a user-defined error that is associated with an element of an AUTOSAR interface. It is specific for the particular functionality or service provided by the AUTOSAR software component.			
Base Class(es)	ARObject, Identifiable			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
errorCode	Integer	1	aggregation	The RTE generator is forced to assign this value to the corresponding error symbol. Note that for error codes certain ranges are predefined (see RTE specification).

Class	EmptySignal			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Class Description	According to the COM Specification, signal groups without signals are allowed. These have a "signalLength" = 0. In this case there shall be an "update-bit" configured.			
Base Class(es)	ARObject			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
systemSignal	SystemSignal	1	reference	Reference to a SystemSignal with "signalLength" = 0 and an UpdateBit.

Class	ClientServerPrimitiveTypeMapping			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Class Description	Mapping of an argument with a primitive datatype to a signal.			
Base Class(es)	ARObject			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
argument	ClientServerPrimitiveType-Mapping_argument	1	reference to instance	Reference to the argument, which ought to be sent over the Communication bus.
systemSignal	SystemSignal	1	reference	Reference to the system signal used to carry the argument

Class	ClientServerCompositeTypeMapping {abstract}
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping
Class Description	Two mappings exist for the composite data types: "ArrayTypeMapping" and "RecordTypeMapping". In both, a primitive datatype will be mapped to a system signal. But it is also possible to combine the arrays and the records, so that an "array" could be an element of a "record" and in the same manner a "record" could be an element of an "array". Nesting these data types is also possible. If an element of a composite data type is again a composite one, the "CompositeTypeMapping" element will be used one more time (aggregation between the ArrayElementMapping and CompositeTypeMapping or aggregation between the RecordElementMapping and CompositeTypeMapping).
Base Class(es)	ARObject

Class	ClientServerArrayTypeMapping			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Class Description	If the compositeType is an Array, the "ArrayTypeMapping" will be used.			
Base Class(es)	ARObject, ClientServerCompositeTypeMapping			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
arrayElementMapping	ClientServerArrayElementMapping	0..*	aggregation	Each ArrayElement must be mapped on a SystemSignal.

Class	ClientServerRecordTypeMapping			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Class Description	If the compositeType is a Record, the "RecordTypeMapping" will be used.			
Base Class(es)	ARObject, ClientServerCompositeTypeMapping			
Attribute	**Datatype**	**Mul.**	**Link Type**	**Attribute Description**
recordElementMapping	ClientServerRecordElement-Mapping	0..*	aggregation	Each RecordElement must be mapped on a SystemSignal.

Class	**ClientServerArrayElementMapping**
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping
Class Description	The ArrayElement may be a primitive one or a composite one. If the element is primitive, it will be mapped to the "SystemSignal" (multiplicity 1). If the element is composite, there will be no mapping to the "SystemSignal" (multiplicity 0). In this case the "ArrayElementMapping" Element will aggregate the "TypeMapping" Element. In that way also the composite datatypes can be mapped to SystemSignals.
Base Class(es)	ARObject

Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
clientServerArrayElementMapping_-arrayElement	ClientServerArrayElement-Mapping_arrayElement	0..1	reference to instance	Reference to an element in an record.
complexTypeMapping	ClientServerCompositeType-Mapping	0..1	aggregation	This aggregation will be used if the element is composite.
signal	SystemSignal	0..1	reference	Reference to the system signal used to carry the primitive ArrayElement.

Class	**ClientServerRecordElementMapping**
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping
Class Description	Mapping of a primitive record element to a SystemSignal. If the element is composite, there will be no mapping (multiplicity 0). In this case the "RecordElementMapping" Element will aggregate the "TypeMapping" Element. In that way also the composite datatypes can be mapped to SystemSignals.
Base Class(es)	ARObject

Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
complexTypeMapping	ClientServerCompositeType-Mapping	0..1	aggregation	This aggregation will be used if the element is composite.
recordElement	ClientServerRecordElement-Mapping_recordElement	0..1	reference to instance	Reference to a element in a record.
signal	SystemSignal	0..1	reference	Reference to the system signal used to carry the primitive RecordElement.

6.2.2 Signal Mapping Constraint (Communication Paths)

One of the tasks of the System Generator is actually to calculate autonomous the communication (signals) between the RTEs and define the needed frames for that communication. These definitions of the frames include implicitly the definition of the paths the AUTOSAR-Signals are transmitted through the system. Thereby the System Generator often has the choice between alternative ways through the system. In the example shown in **Figure 37** the System Generator would have the choice between two ways (Path1: CAN3 or Path2: CAN1-GW-CAN2) for a signal from ECU2 to ECU4. If no further information is given the decision will be made e.g. by means of boundary conditions like busload, transmissions speed, etc.
Signal Mapping Constraints allow to further restrict or specify the path(s) a signal is allowed to be transmitted over. A path is specified by an ordered list of ECUCommunicationPortInstance references. The System Generator needs to assert that for the chosen path each of these ECUCommunicationPortInstances is being connected to the next one within a CommunicationCluster, and the Signal to be transmitted is covered by a ClusterSignal for each CommunicationCluster being part of the Path.

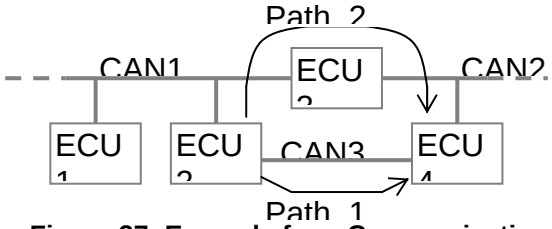

Figure 37: Example for a Communication Path

Usage of the Template to describe the "System Constraints"
However it can be necessary e.g. for optimization and safety reasons to make additional guidelines for the System Generator, which specific way a signal between two Software Components should take in the network without defining in which frame and with which timing it is transmitted.
There exist four different constraints for signals regarding the signal path:

- The `CommonSignalPath` describes that two signals must take the same way (Signal Path) in the topology.
- The `ForbiddenSignalPath` describes the way (Signal Path) that a signal must not take in the topology, e.g. in case of safety critical transmission.
- The `PermissibleSignalPath` describes the way (Signal Path) a signal can take in the topology. If more than one PermissibleSignalPath is defined for the same signal/operation attributes, any of them can be chosen.
- The `SeparateSignalPath` describes that two or more signals must not take the same way (Signal Path) in the topology e.g. in case of redundant transmission. It is also possible that the same signal is aggregated two times by the `SeparateSignalPath` element to indicate that this signal should be transmitted redundantly over two different paths.

The meta-model part, which describes the Communication Path constraints, will be explained in the following sections.

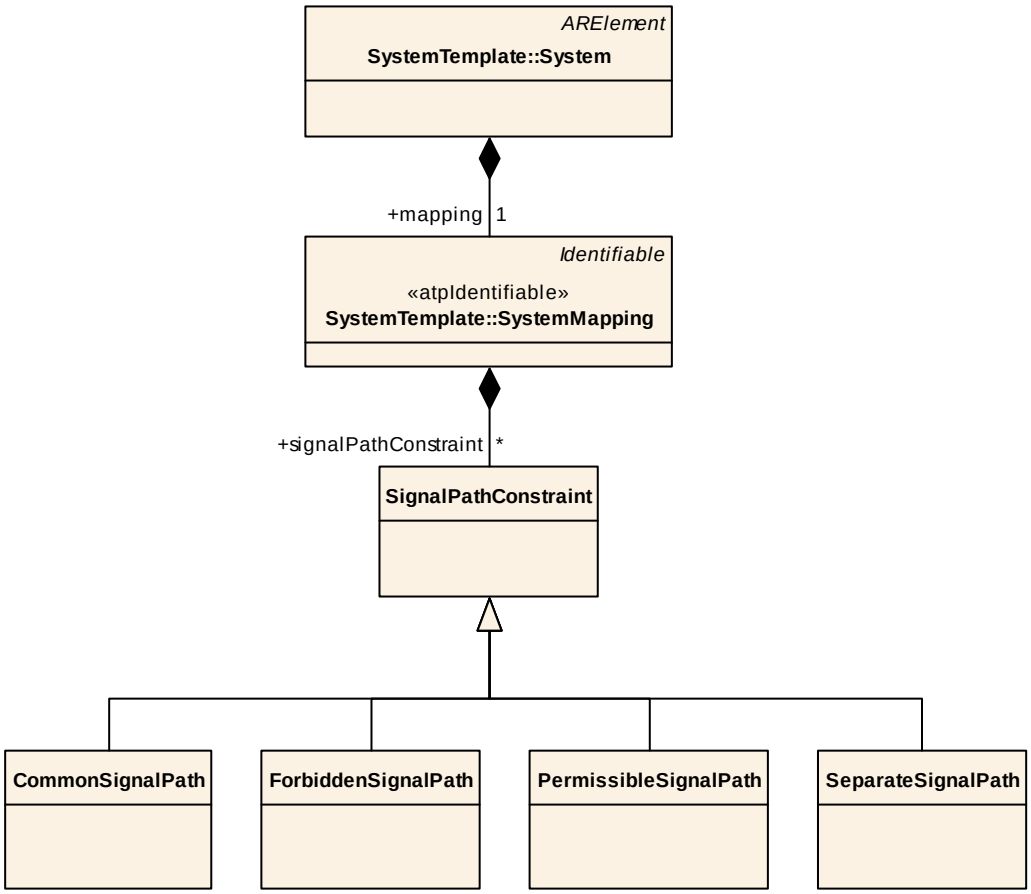

Figure 38: Communication Path Description (SignalPathConstraints)

6.2.2.1 CommonSignalPath

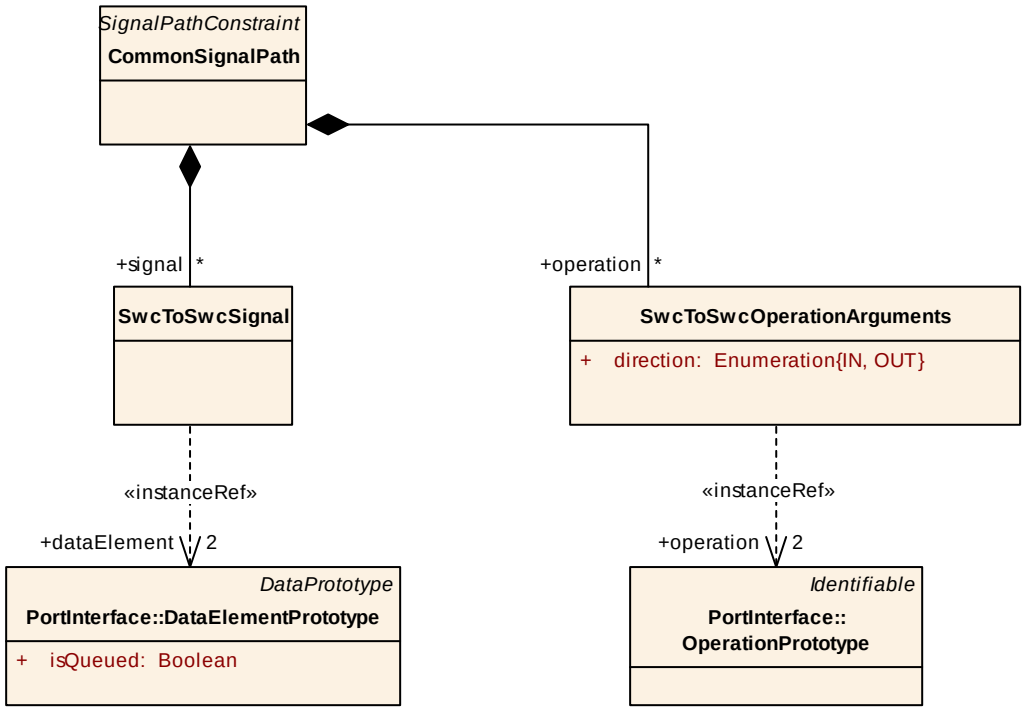

Figure 39: Description of signals that must take the same way in the topology (CommonSignalPath)

Class	**CommonSignalPath**			
Package	M2::AUTOSARTemplates::SystemTemplate::SignalPaths			
Class Description	The CommonSignalPath describes that two or more SwcToSwcSignals and/or SwcToSwcOperationArguments must take the same way (Signal Path) in the topology.			
Base Class(es)	ARObject, SignalPathConstraint			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
operation	SwcToSwcOperationArguments	0..*	aggregation	
signal	SwcToSwcSignal	0..*	aggregation	The SwcToSwcSignals that must take the same way (Signal Path) in the topology.

Class	**SwcToSwcSignal**			
Package	M2::AUTOSARTemplates::SystemTemplate::SignalPaths			
Class Description	The SwcToSwcSignal describes the information (data element) that is exchanged between two SW Components. On the SWC Level it is possible that a SW Component sends one data element from one P-Port to two different SW Components (1:n Communication). The SwcToSwcSignal describes exactly the information which is exchanged between one P-Port of a SW Component and one R-Port of another SW Component.			
Base Class(es)	ARObject			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
dataElement	SwcToSwcSignal_dataElement	2	reference to instance	Reference to a data element on the PPort and to the same data element on the RPort.

6.2.2.2 ForbiddenSignalPath

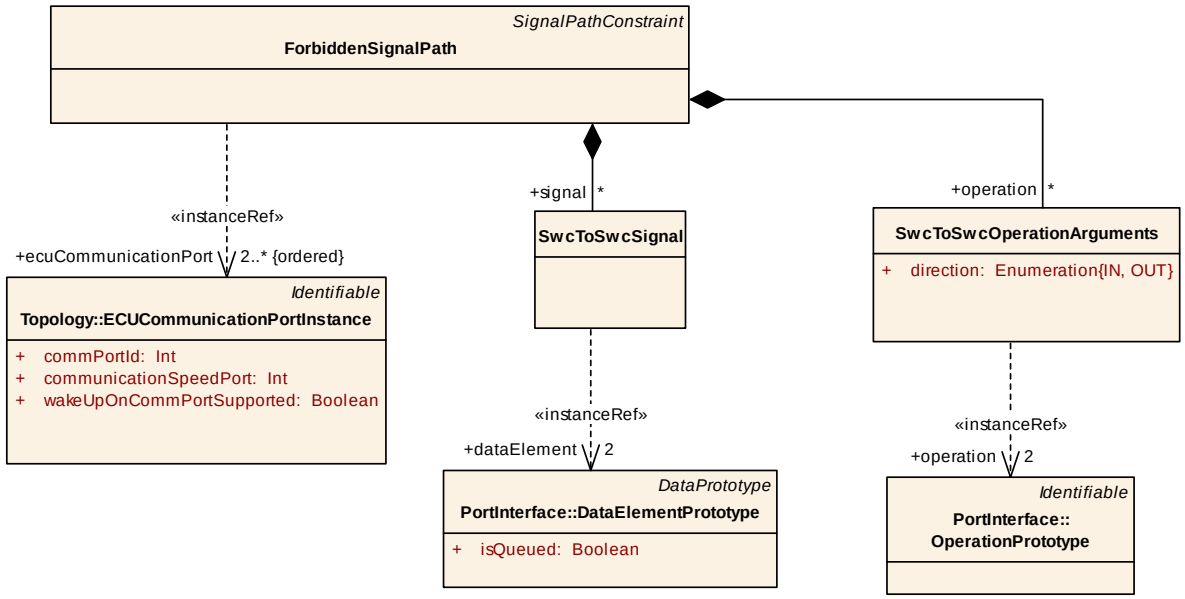

Figure 40: Description of the signal path that a signal must not take in the topology (ForbiddenSignalPath)

Class	**ForbiddenSignalPath**			
Package	M2::AUTOSARTemplates::SystemTemplate::SignalPaths			
Class Description	The ForbiddenSignalPath describes the way (Signal Path) which an element must not take in the topology. Such a signal path can be a constraint for the communication matrix, because such a path has an effect on the frame generation and the frame path.			
Base Class(es)	ARObject, SignalPathConstraint			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
ecuCommunicationPort	ForbiddenSignalPath_ecuCommunication-Port	2..*	reference to instance	A Signal path is represented by a reference to a sequence of HW Ports.
operation	SwcToSwcOperationArguments	0..*	aggregation	Reference to the operation arguments of one operation which must not take the predefined way in the topology.
signal	SwcToSwcSignal	0..*	aggregation	The data element which must not take the predefined way in the topology.

The following figure shows an example for a forbidden signal path. The data element must not take the path between controller 1 of ECU 1 and controller 3 of ECU2. The path between the communication controller 2 and communication controller 3 is still allowed. In order to describe, that also the path between controller 2 and controller 3 is forbidden, an additionally forbidden signal path must be described.

Figure 41: Example for a forbidden Signal Path

6.2.2.3 PermissibleSignalPath

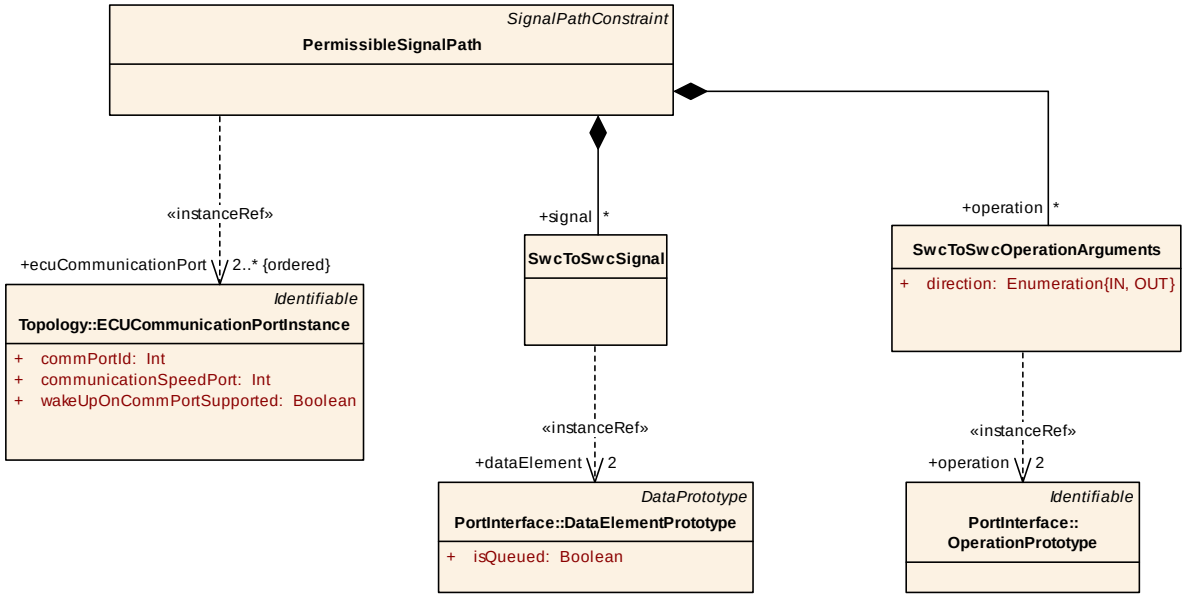

Figure 42: Description of the signal path that a signal must take in the topology (PermissibleSignalPath)

Class	**PermissibleSignalPath**			
Package	M2::AUTOSARTemplates::SystemTemplate::SignalPaths			
Class Description	The PermissibleSignalPath describes the way (Signal Path) a data element can take in the topology. If more than one PermissibleSignalPath is defined for the same signal/operation attributes, any of them can be chosen. Such a signal path can be a constraint for the communication matrix . This path describes that one data element should take path A and not path B. This has an effect on the frame generation and the frame path.			
Base Class(es)	ARObject, SignalPathConstraint			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
ecuCommunicationPort	PermissibleSignalPath_ecu-CommunicationPort	2..*	reference to instance	A Signal path is represented by a reference to a sequence of HW Ports. The HW Ports should be indicated in the order in which they will be passed by the data element.
operation	SwcToSwcOperationArguments	0..*	aggregation	The arguments of an operation that can take the predefined way in the topology.
signal	SwcToSwcSignal	0..*	aggregation	The data element which can take the predefined way in the topology.

6.2.2.4 SeparateSignalPath

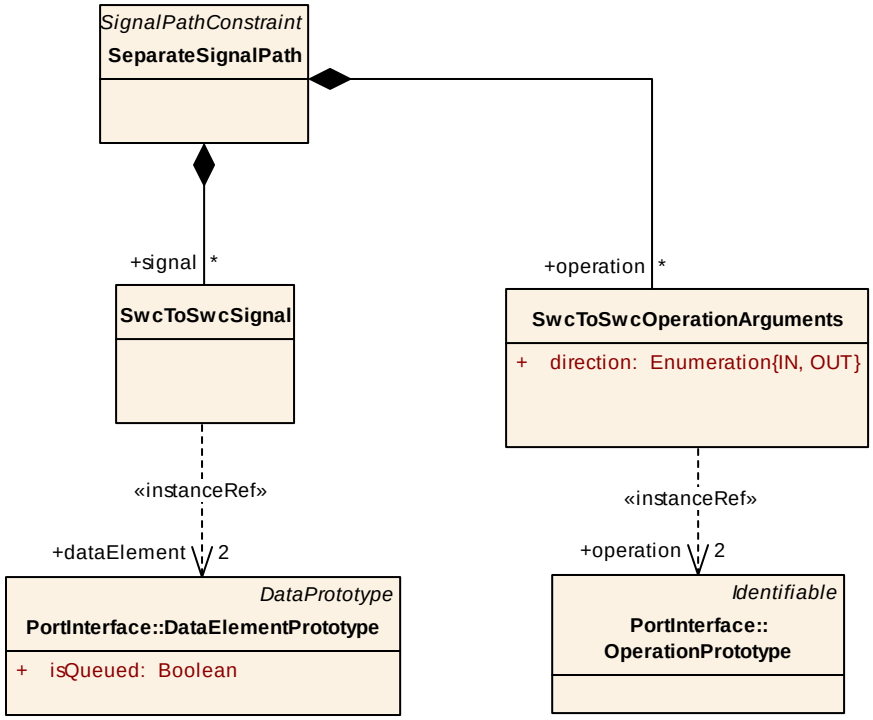

Figure 43: Description of signals that must not take the same way in the topology (SeparateSignalPath)

Class	**SeparateSignalPath**			
Package	M2::AUTOSARTemplates::SystemTemplate::SignalPaths			
Class Description	The SeparateSignalPath describes that two SwcToSwcSignals and/or SwcToSwcOperationArguments must not take the same way (Signal Path) in the topology (e.g. Redundancy).			
Base Class(es)	ARObject, SignalPathConstraint			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
operation	SwcToSwcOperationArguments	0..*	aggregation	The SwcToSwcOperationArguments that must not take the same way (Signal Path) in the topology.
signal	SwcToSwcSignal	0..*	aggregation	The SwcToSwcSignals that must not take the same way (Signal Path) in the topology.

Usage of the Template to describe the “System Configuration”

Signal paths are not an obligatory part of the System Configuration Description. In the final System Configuration Description every signal is assigned to a frame. Thereby the paths of the AUTOSAR-Signals are implicitly described.
But that does not mean that signal path information have to be deleted after the system generation step. By deleting the signal paths you would lose the information why you have chosen e.g. a specific frame for a signal. If you extend or change the system at a later date the missing information about signal paths could lead to a not wanted signal mapping if the system Generator remaps the signals.

6.3 RTE and basic software resource estimations

Important constraints for system partitioning are the available resources on the ECUs in the system. For SW components, the resource estimations can be stated in SW component descriptions. It is however not only SW components that require resources. AUTOSAR RTE and basic software running on the ECU have resource needs as well.
The realization of the RTE and the kind of basic software to be run on a certain ECU depend on the implicit and explicit usage of all basic software by the software components. The software components need to communicate internally and with software components on other ECUs. Furthermore, they have different needs with respect to scheduling. This results in implicit use of e.g. communication and operating system software. In addition, the software components make explicit use of basic software when they e.g. utilize system services (e.g. diagnostics) and access sensors/actuators via the I/O abstraction layer or the complex device driver abstraction layer. Thus, the resource consumption of the RTE and the basic software depend on the SW Components mapped to the ECU, since this determines the exact configuration of the RTE and the basic software.
The resource consumption for RTE and basic software are specified using class EcuResourceEstimation. Each estimation is performed for a specific ECU (references by EcuResourceEstimation) and for a specific set of SW mapped to that ECU (also referenced from EcuResourceEstimation). Different resource estimations for a specific ECU, but with different mappings may exist, e.g. for different variants of the system, or to show the difference of resource needs for different mappings.
The following figure shows the meta-model for resource estimations for RTE and basic SW.

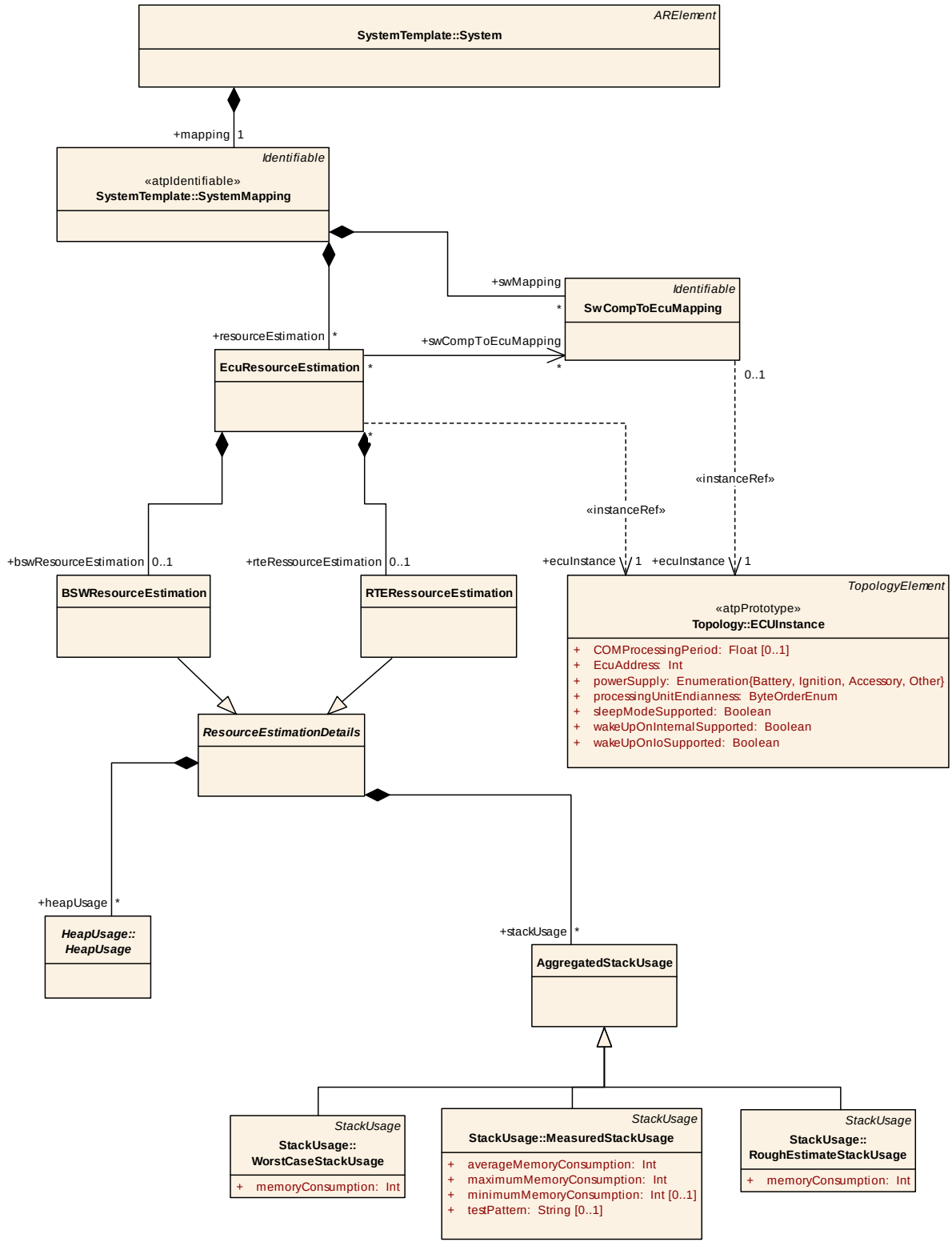

Figure 44: ECU resource estimations (ResourceEstimation)

Class	**EcuResourceEstimation**			
Package	M2::AUTOSARTemplates::SystemTemplate::SWmapping			
Class Description	Resource estimations for RTE and BSW of a single ECU instance,			
Base Class(es)	ARObject			
Attribute	*Datatype*	*Mul.*	*Link Type*	*Attribute Description*
bswResourceEstimation	BSWResourceEstimation	0..1	aggregation	Estimation for the resource consumption of the basic software.
ecuInstance	EcuResourceEstimation_ecu-Instance	1	reference to instance	Reference to the ECU this estimation is done for.
rteRessourceEstimation	RTERessourceEstimation	0..1	aggregation	Estimation for the resource consumption of the run time environment.
swCompToEcuMapping	SwCompToEcuMapping	0..*	reference	References to SwCompToEcuMappings that have been taken into account for the resource estimations. This way it is possible to define dfferent EcuResourceEstimations with different mappings, e.g. before and after mapping an additional SW component.

Class	**BSWResourceEstimation**
Package	M2::AUTOSARTemplates::SystemTemplate::SWmapping
Class Description	Estimations for the resource consumption for the basic software.
Base Class(es)	ARObject, ResourceEstimationDetails

Class	**RTERessourceEstimation**
Package	M2::AUTOSARTemplates::SystemTemplate::SWmapping
Class Description	Estimations for the resource consumption for the AUTOSAR run time environment.
Base Class(es)	ARObject, ResourceEstimationDetails

7 ECU Extract of the System Configuration Description

As shown in **Figure 2** only parts of the System Configuration Description are used as input for the ECU generation step for an individual ECU. Only the relevant information that is needed to generate this individual ECU is included, all other information is removed from the configuration description. This extract is called *ECU extract of the System Configuration Description*.
In general, from a given System Configuration Description, it is straightforward to generate an extract for a specific ECU: Take the XML input and remove all elements that are not relevant for that ECU, such as SW components mapped to other ECUs, topology information elements that are not directly connected to the ECU, etc.
There is one exemption to this simple “remove” rule: the communication mapping may need to be extended, which will be described in more detail below.

7.1 Inclusion of elements

The following table shows the rules that define whether an element has to be included in the ECU extract or not:

System top level	
System	Always included
SystemCommunication	Always included
SystemMapping	Always included
SystemTopologyInstance	Always included
SoftwareComposition	Always included
GatewayInstance	Included if the referenced GatewayType is included
CommunicationMatrixInstance	Included if the referenced CommunicationMatrixType is included
Topology	
SystemTopologyType	Always included (the one referenced by SystemTopologyInstance of the system under consideration)
CommunicationCluster	Included if ECU is connected to that cluster
Physical Channel	Included if ECU is connected to that physical channel
Topology Element	Not included
Hub	Not included
PhysicalMediumSegment	Included if ECU is connected to that physical medium segment
ECUCommunicationPortInstance	Included if part of ECU under consideration
Connector	Included if part of ECU under consideration
ECUInstance	Included if ECU under consideration
SystemSignal	

SystemSignal	Included if there is a PDUType included that references to this SystemSignal
ClusterSignal	Included if the aggregating PDUPrototype is included
ClusterSignalReceiver	Included if the aggregating ClusterSignal is included
ClusterSignalTransmitter	Included if the aggregating ClusterSignal is included
SystemSignalGroup	Included if the referenced SystemSignals are included
Communication	
CommunicationMatrixType	Included if the CommunicationCluster referenced is included and if there is a CommunicationMatrixInstance defined in the system that references to this element.
Frame	included if ECU sends or receives this frame, may also be included if aggregating physical channel is included (if e.g. needed for rest bus simulation and filter settings in ECU)
PDUPrototype	Included if the aggregating Frame is included
Timing	included if aggregating Frame is included. Note that timings aggregated by SystemSignals are only requirements and don't need to be incuded.
SendCondition	included if aggregating timing is included and if aggregating frame is sent by ECU
ActiveCondition	included if aggregating timing is included and if aggregating frame is sent by ECU
StartCondition	included if aggregating timing is included and if aggregating frame is sent by ECU
StopCondition	included if aggregating timing is included and if aggregating frame is sent by ECU
SystemStateTrigger	included if aggregating timing is included and if aggregating frame is sent by ECU
SignalTrigger	included if aggregating timing is included and if aggregating frame is sent by ECU
FreeTextCondition	included if aggregating timing is included and if aggregating frame is sent by ECU
LinSchedulingTable	included if aggregating CommunicationMatrix is included
LinSchedulingTableEntry	included if aggregating LinSchedulingTable is included AND (ECU is Lin Master OR LinFrame referenced is sent or received by ECU)
GatewayType	included if ECU under consideration is referenced and if there is a

	GatewayInstance referencing that is part of the system under consideration
GatewayEntry	included if aggregating ECU is included
CommunicationProtocols	
CommProtocolInstance	included if aggregated CommProtocolSignalRole CommProtocolFrameRole or CommProtocolSignalGroupRole needs to be included
CommProtocolType	included if any referencing CommProtocolInstance is included
CommProtocolElement	included if aggregating CommProtocolType is included
CommProtocolSignalRole	included if Frame that carries the referenced SystemSignal is sent or received by ECU
CommProtocolFrameRole	included if referenced Frame is sent or received by ECU
CommProtocolSignalGroupRole	included if Frame that carries SystemSignals from the SignalGroup is sent or received by ECU
PDUStructure	
PDUType	included if any referencing PDUPrototype is included
SignalPosition	included if aggregating PDUType is included
Multiplexer	included if aggregating PDUType is included
SubPDU	included if aggregating Multiplexer is included
DataMapping	
DataMapping	Always included
ClientServerToSignalGroupMapping	Added or included if a signal, in which an argument of an operation is transported, is sent or received by ECU. Added means that the mapping may need to be added if only a mapping of the sender existed and ECU is receiver. Then the corresponding receiving SW component's port needs to be mapped.
ClientServerPrimitiveTypeMapping	included if aggregating ClientServerToSignalGroupMapping is included
CompositeTypeMapping	Included if aggregating ClientServerToSignalGroupMapping is included OR

	included if aggregating SenderReceiverTypeMapping is included
RecordElementMapping	Included if aggregating CompositeTypeMapping is included
ArrayElementMapping	Included if aggregating CompositeTypeMapping is included
SenderReceiverToSignalGroupMapping	Added or included if a signal, which is part of a signal group, is sent or received by ECU. Added means that the mapping may need to be added if only a mapping of the sender existed and ECU is receiver. Then the corresponding receiving SW component's port needs to be mapped.
SenderReceiverToSignalMapping	Added or included if signal is sent or received by ECU. Added means that the mapping may need to be added if only a mapping of the sender existed and ECU is receiver. Then the corresponding receiving SW component's port needs to be mapped.
SW Mapping	
EcuResourceEstimation	Included if ECU is referenced.
SwCompToEcuMapping	Included if ECU is referenced.
SwCompToImplMapping	Included if SWC, which is mapped to the ECU, is referenced.
MappingConstraint	Not included (also all aggregated elements are not included)
BSWResourceEstimation	Included if EcuResourceEstimation is included
RTEResourceEstimation	Included if EcuResourceEstimation is included
SignalPath	
SignalPathConstraint	Not included (also all aggregated elements are not included)
SwcToSwcSignal	Not included
SwcToSwcOperationArguments	Not included
From SW Component Template	
CompositionType	Included if it is the flattened top level composition of the system. Aggregated elements are included if they are mapped to this ECU, see below.
ComponentPrototype and the matching type	Included if mapped to this ECU, i.e. referenced by a SwCompToEcuMapping

	that references to ECU under consideration
Implementation and all aggregated elements	Included if mapped to this ECU, i.e. referenced by a SwCompToImplMapping that references to a SWC, which is mapped to ECU under consideration
Internal Behavior	Included if at least one ComponentPrototype of the referenced AtomicSoftwareComponentType is mapped to this ECU
Data types etc.	
From ECU Resource Template	
ECU and everything aggregated	included if referencing ECU instance is included (i.e. ECU is of this type)

7.2 SW component inclusion and data mapping

As mentioned before, there is a slight complication to above include/exclude rules. This can be shown best with an example. Assume a simple topology with two ECUs A and B and two frames X (sent from A to B) and Y (sent from B to A) as shown in **Figure 45**.

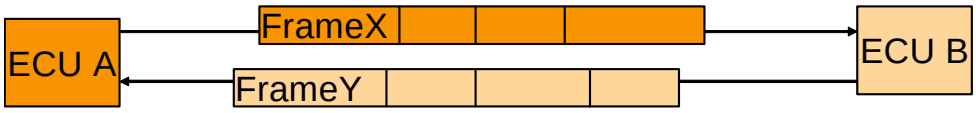

Figure 45: Example topology with two ECUs and two frames exchanged between them

Furthermore assume a SW composition as shown in **Figure 46**. It consists of five atomic SW components ‘A1’ to ‘A3’ (aggregated in composition ‘SwCompA’) and ‘B1’ / ‘B2’. (aggregated in composition ‘SWCompB’). The overall composition ‘SWCompAplusB’ aggregate ‘SwCompA’ and ‘SWCompB’.

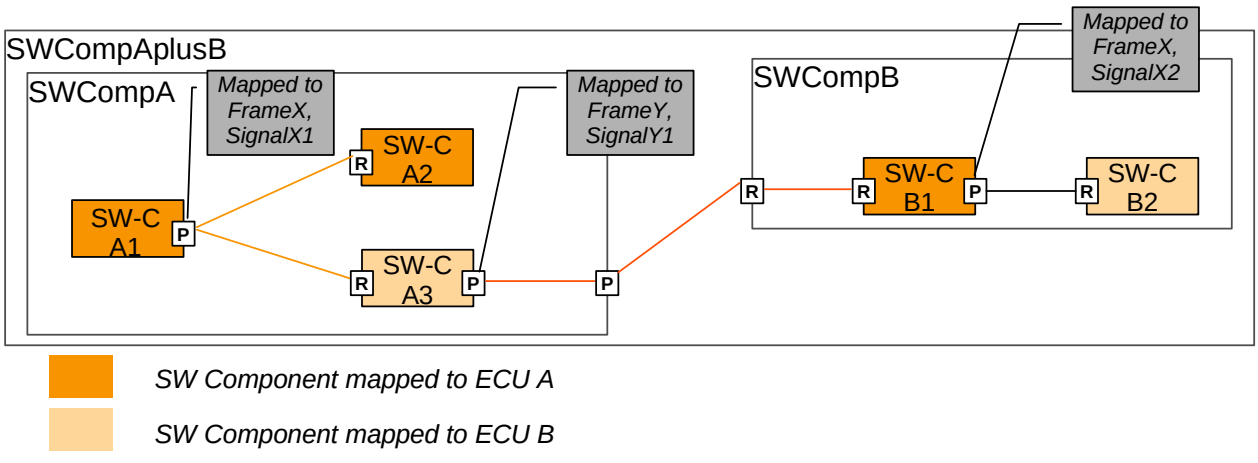

Figure 46: Example SW composition with mapping information

The atomic SW components ‘A1’, ‘A2’ and ‘B1’ are mapped to ‘ECU A’, the others to ‘ECU B’. The data sent from
‘A1’ to ‘A3’ is mapped to ‘FrameX’, ‘SignalX1’,
‘B1’ to ‘B2’ is mapped to ‘FrameX’, ‘SignalX2’ and
‘A3’ to ‘B1’ is mapped to ‘FrameY’, ‘SignalY1’.
As usual, the data mapping rules refer to the data element in the P-Port of the sending SW component.

Figure 47 shows how the ECU extract for ECU A of this SW composition would look like:
Only those atomic SW components are included that are mapped to ECU A.
All compositions are included since they have aggregated atomic SW components which are included and cannot be left out for that reason.
Only those connectors are included that represent intra-ECU communication (in our example, only ‘A1’ to ‘A2’), since this information is still needed for the RTE generation.
Connectors that were used to connect to SW components that are not included in the extract are not included either. Instead, the mapping to a signal in a frame is used to identify the source/destination of that data.
Furthermore, the relevant topology information and communication matrix have to be included, but they are out of scope of this example.

The problem that new mapping rules have to be added arises with the mapping to ‘FrameY’, ‘SignalY1’: Since SW component ‘A3’, which was referenced in the original mapping, is no longer included, the data mapping needs a new data element in a port to reference to. In the example, it is the required port of ‘B1’, so that the ECU generator has the information that B1 receives the data via ‘FrameY’.

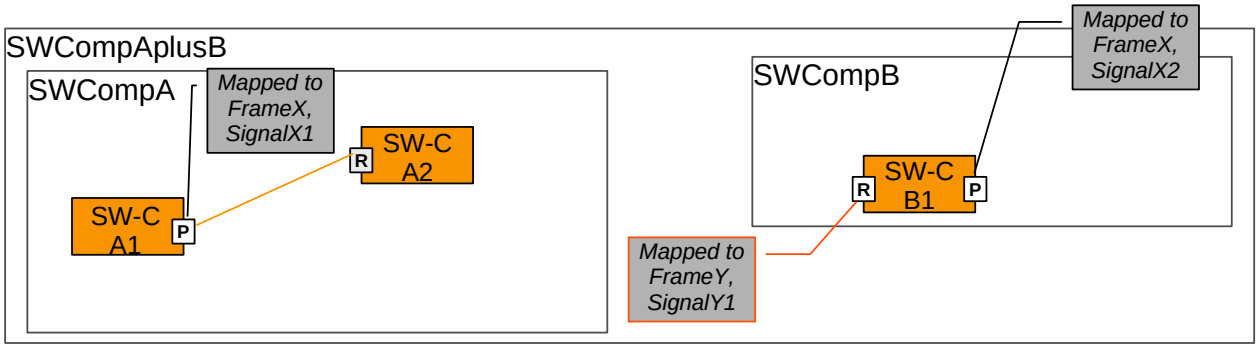

Figure 47: Example ECU extract for ECU A of above introduced composition

8 Reference

8.1 References to AUTOSAR documents

[1] AUTOSAR Glossary
https://svn.autosar.org/repos/10Releases/
AUTOSAR_Glossary.pdf
[2] Meta Model
https://svn.autosar.org/repos/10Releases/
AUTOSAR_Metamodel.eap
[3] Template Modeling Patterns,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_TemplateModelingPatterns.pdf.
[4] Model Persistence Rules for XML,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_ModelPersistenceRulesforXML.pdf
[5] Template UML Profile and Modeling Guide
https://svn.autosar.org/repos/10Releases/
AUTOSAR_TemplateModelingGuide.pdf
[6] AUTOSAR Methodology
https://svn.autosar.org/repos/10Releases/
AUTOSAR_Methodology.pdf
[7] AUTOSAR Main Requirement
https://svn.autosar.org/repos/10Releases/
AUTOSAR_MainRequirements.pdf
[8] Specification of the Virtual Functional
https://svn.autosar.org/repos/10Releases/
AUTOSAR_Spec_of_VFB.pdf
[9] Software Component Template
https://svn.autosar.org/repos/10Releases//
AUTOSAR_SoftwareComponentTemplate.pdf
[10] Design Specification for the ECU Resource Template
https://svn.autosar.org/repos/10Releases/
AUTOSAR_ResourceTemplateECU.pdf
[11] Specification of RTE Software
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_RTE.pdf
[12] Specification of Communication
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_COM.pdf

8.2 References to external documents

[13] Fibex –Field Bus Exchange Format, Version 2.0, http://www.asam.net

[14] FlexRay Protocol Specification, http://www.flexray-group.com

[15] LIN Specification Package, http://www.lin-subbus.org/

[16] MSR Sub-project MEDOC, Structure Principles, Scope Networks, http://www.msr-wg.de/

[17] MOST Cooperation: MOST Specification, http://www.mostnet.de/

9 Annex

9.1 Abbreviation list

Abbreviation	*Meaning*
CAN	Controller Area Network
DTD	Document Type Definition
I²C	Inter-Integrated Circuit
LIN	Local Interconnect Network
MOST	Media Oriented Systems Transport
RTE	Runtime Environment
SPI	Serial Peripheral Interface
SWC	Software Component
SWC-T	Software Component Template
SYS-T	System Template
UML	Unified Modeling Language
XML	Extensible Markup Language
VFB	Virtual Functional Bus
ID	Identifier
ISG	Inter-slot Gap
CC	Communication Controller
NAD	Node Address for Diagnostic
OBD	Onboard Diagnostic
ECU	Electrical Control Unit
SID	Service Identifier
NIT	Network Idle Time
CAS	Collision Avoidance Symbol
PDU	Protocol Data Unit
POC	Protocol Operation Control
FIBEX	Field Bus Exchange Format

9.2 Comparison with other description formats

There are further XML description formats which describe the Communication Matrices. The next sections give a short survey of FIBEX and MSR Net. Both description formats had an influence on this document.

9.2.1 Fibex

FIBEX (Field Bus Exchange Format) describes an XML exchange format proposed for data exchange between tools that deal with message-oriented bus communication Systems.
The format supports the most common automotive data buses: LIN, CAN, MOST, FlexRay, byteflight. The covered areas of the exchange format are the functional network, system topology and the communication level. The functional network describes the software architecture of the system. In the system topology the logical

layout of the system is described. This means it is documented which ECU is connected to which bus. The central purpose of a communication system is the exchange of frames with certain properties. The format is able to describe frames and their timing properties.
The XML structure presented in this document is affected by FIBEX. However, it was impossible to take over the complete contents from FIBEX. Some adaptations had to be made. There are some general reasons why FIBEX cannot be used completely:

- System Template needs to interact with the other AUTOSAR descriptions. E.g. Functions in FIBEX are SW components in AUTOSAR.
- Referencing between elements is solved differently in FIBEX and AUTOSAR.
- Focus is slightly different: System Template tries to define everything that needs to be known at system level about the topology, communication matrix etc. ECU Internals are left to the ECU description (e.g. concrete mapping of frames to a CONNECTOR/CommunicationPort is not done in System Template, since this is information that may be hidden for other ECUs).
- Naming of elements: due to different philosophies in naming, the terminology used is not always aligned. The System Template might be able to align more in many cases, but there are reasons why it is not always possible.
- Names used in other parts of the AUTOSAR MetaModel may not be reused.
- Class names must be unique in AUTOSAR

9.2.2 MSR MEDOC for Networks

MSR/MEDOC is a subtask of the MSR consortium. MEDOC is an acronym for MSR Engineering Data Objects and Contents. MEDOC develops methods, standards and implementation for information exchange in the engineering process. The results of MSR/MEDOC are the defined structures, modeled as XML-DTDs. One of these structures is MSR/MEDOC Networks. The objective of MEDOC Networks is to define one format for all possible bus descriptions and descriptions for networks. Currently the description is only for CAN. The transfer to further network formats will occur in the future. The MSR Net describes the Network Topology and how the information is transported on the network. Furthermore, it describes how the signals are packed into messages. FIBEX is the descendant of MSR Medoc. So FIBEX was mainly considered in this Document with regard to the contents.