

|                                |                         |
|--------------------------------|-------------------------|
| <b>Document Title</b>          | UML Profile for AUTOSAR |
| <b>Document Owner</b>          | AUTOSAR GbR             |
| <b>Document Responsibility</b> | AUTOSAR GbR             |
| <b>Document Version</b>        | 1.0.1                   |
| <b>Document Status</b>         | Final                   |

| <b>Document Change History</b> |                |                           |                           |
|--------------------------------|----------------|---------------------------|---------------------------|
| <b>Date</b>                    | <b>Version</b> | <b>Changed by</b>         | <b>Change Description</b> |
| 27.06.2006                     | 1.0.1          | AUTOSAR<br>Administration | Layout Adaptations        |
| 28.04.2006                     | 1.0.0          | AUTOSAR<br>Administration | Initial release           |

## Disclaimer

This specification as released by the AUTOSAR Development Partnership is intended **for the purpose of information only**. The use of material contained in this specification 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 this Specification.

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.

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Copyright © 2004-2006 AUTOSAR Development Partnership. All rights reserved.

## 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 AUTOSAR compliance certification 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.

## Abstract

This document describes a UML 2.0 Profile for AUTOSAR, initially covering the V2.0 AUTOSAR basic software feature list.

## Table of Contents

|         |  |    |
|---------|--|----|
| 1.      | Introduction.....  | 8  |
| 1.1.    | Goals.....   | 8  |
| 1.2.    | Scope.....   | 8  |
| 1.3.    | Terminology .....  | 8  |
| 2.      | Example use of the Profile.....                                  | 10 |
| 3.      | Software Component Model .....                                   | 14 |
| 3.1.    | Compositions .....   | 14 |
| 3.1.1.  | ComponentType (from Components).....                             | 14 |
| 3.1.2.  | ComponentPrototype (from Composition).....                       | 15 |
| 3.1.3.  | CompositionType (from Composition).....                          | 16 |
| 3.1.4.  | AtomicSoftwareComponentType (from Components).....               | 16 |
| 3.1.5.  | SensorActuatorSoftwareComponentType (from Components) .....      | 17 |
| 3.2.    | Ports and Connectors.....  | 18 |
| 3.2.1.  | PortPrototype (from Components) .....                            | 19 |
| 3.2.2.  | PPortPrototype (from Components) .....                           | 20 |
| 3.2.3.  | ServerPort.....  | 20 |
| 3.2.4.  | ReceiverPort .....   | 21 |
| 3.2.5.  | RPortPrototype (from Components).....                            | 21 |
| 3.2.6.  | ClientPort .....   | 22 |
| 3.2.7.  | SenderPort.....  | 22 |
| 3.2.8.  | ConnectorPrototype (from Composition).....                       | 22 |
| 3.2.9.  | AssemblyConnectorPrototype (from Composition) .....              | 23 |
| 3.2.10. | DelegationConnectorPrototype (from Composition) .....            | 24 |
| 3.3.    | Interfaces .....   | 25 |
| 3.3.1.  | PortInterface (from PortInterface).....                          | 26 |
| 3.3.2.  | SenderReceiverInterface (from PortInterface).....                | 27 |
| 3.3.3.  | DataPrototype (from Datatype::Datatypes) .....                   | 28 |
| 3.3.4.  | DataElementPrototype (from PortInterface) .....                  | 28 |
| 3.3.5.  | ClientServerInterface (from PortInterface) .....                 | 28 |
| 3.3.6.  | OperationPrototype (from PortInterface) .....                    | 29 |
| 3.3.7.  | ArgumentPrototype (from PortInterface) .....                     | 29 |
| 3.4.    | Data Types.....  | 30 |
| 3.4.1.  | PrimitiveType (from Datatype::Datatypes) .....                   | 31 |
| 3.4.2.  | Range (from Datatype::Datatypes).....                            | 32 |
| 3.4.3.  | BooleanType (from Datatype::Datatypes).....                      | 32 |
| 3.4.4.  | IntegerType (from Datatype::Datatypes) .....                     | 33 |
| 3.4.5.  | RealType (from Datatype::Datatypes).....                         | 33 |
| 3.4.6.  | OpaqueType (from Datatype::Datatypes) .....                      | 33 |
| 3.4.7.  | PrimitiveTypeWithSemantics (from Datatype::Datatypes).....       | 34 |
| 3.4.8.  | EnumerationType.....   | 34 |
| 3.5.    | Internal Behavior .....  | 35 |
| 3.5.1.  | InternalBehavior (from InternalBehavior) .....                   | 35 |
| 3.5.2.  | Implementation (from InternalBehavior:: Implementation).....     | 36 |
| 3.5.3.  | ResourceConsumption (from InternalBehavior::Implementation)..... | 37 |
| 3.5.4.  | Code (from InternalBehavior::Implementation).....                | 37 |

|         |   |    |
|---------|---|----|
| 3.5.5.  | The Runnable Entities.....  | 38 |
| 3.5.6.  | Data access .....   | 40 |
| 3.5.7.  | RTE Events.....   | 45 |
| 4.      | ECU Model .....   | 50 |
| 4.1.    | ECUResourceTemplate – General Elements.....                               | 50 |
| 4.1.1.  | HWElement (from ECUResourceTemplate).....                                 | 50 |
| 4.1.2.  | HWElementContainer (from ECUResourceTemplate) .....                       | 51 |
| 4.1.3.  | ECU (from ECUResourceTemplate) .....                                      | 51 |
| 4.1.4.  | HWPort (from ECUResourceTemplate) .....                                   | 52 |
| 4.1.5.  | AssemblyHWConnection (from ECUResourceTemplate) .....                     | 53 |
| 4.2.    | Sensor Actuator.....  | 54 |
| 4.2.1.  | SensorActuatorHW (from ECUResourceTemplate::Sensor Actuator). 54          |    |
| 4.2.2.  | ActuatorHW (from ECUResourceTemplate::Sensor Actuator).....               | 54 |
| 4.2.3.  | SensorHW (from ECUResourceTemplate::Sensor Actuator).....                 | 54 |
| 4.2.4.  | DisplayHW (from ECUResourceTemplate::Sensor Actuator).....                | 54 |
| 4.3.    | Memory .....  | 55 |
| 4.3.1.  | MemoryMappedHWPort (from ECUResourceTemplate::Memory).....                | 55 |
| 4.3.2.  | MemoryMappedAssemblyHWConnection (from ECUResourceTemplate::Memory) ..... | 55 |
| 4.3.3.  | ProvidedMemorySegment (from ECUResourceTemplate::Memory)... 55            |    |
| 4.3.4.  | ProvidedNVMemorySegment (from ECUResourceTemplate::Memory)..              |    |
|         | .....   | 56 |
| 4.4.    | Basic Elements.....   | 56 |
| 4.4.1.  | ErrorDetectionCorrection (from ECUResourceTemplate::Basic Elements) ..... | 56 |
| 4.5.    | ECU Electronics .....   | 56 |
| 4.5.1.  | ECUElectronics (from ECUResourceTemplate::ECU Electronics).....           | 56 |
| 4.5.2.  | Clock (from ECUResourceTemplate::ECU Electronics).....                    | 57 |
| 4.6.    | Peripherals .....   | 58 |
| 4.6.1.  | Peripheral (from ECUResourceTemplate::Peripherals) .....                  | 58 |
| 4.6.2.  | AnalogueIO (from ECUResourceTemplate::Peripherals).....                   | 58 |
| 4.6.3.  | ADC (from ECUResourceTemplate::Peripherals) .....                         | 59 |
| 4.6.4.  | DAC (from ECUResourceTemplate::Peripherals) .....                         | 59 |
| 4.6.5.  | DigitalIO (from ECUResourceTemplate::Peripherals).....                    | 59 |
| 4.6.6.  | CommunicationPeripheral (from ECUResourceTemplate::Peripherals)..         |    |
|         | .....   | 60 |
| 4.6.7.  | CommunicationProtocol (from ECUResourceTemplate::Peripherals). 60         |    |
| 4.6.8.  | CommunicationFilter (from ECUResourceTemplate::Peripherals) .....         | 60 |
| 4.6.9.  | PulseWidthPeripheral (from ECUResourceTemplate::Peripherals) ... 61       |    |
| 4.6.10. | PWD (from ECUResourceTemplate::Peripherals).....                          | 61 |
| 4.6.11. | PWM (from ECUResourceTemplate::Peripherals) .....                         | 61 |
| 4.6.12. | Buffer (from ECUResourceTemplate::Peripherals).....                       | 62 |
| 4.6.13. | PeripheralHWPort (from ECUResourceTemplate::Peripherals) .....            | 62 |
| 4.7.    | ProcessingUnit .....  | 62 |
| 4.7.1.  | ProcessingUnit (from ECUResourceTemplate::Processing Unit).....           | 62 |
| 5.      | System Modeling.....  | 63 |
| 5.1.    | Top Level .....   | 64 |
| 5.1.1.  | System (from SystemTemplate).....   | 65 |

|         |  |    |
|---------|--|----|
| 5.1.2.  | SystemTopologyInstance (from SystemTemplate).....  | 65 |
| 5.1.3.  | SystemCommunication (from SystemTemplate) .....  | 65 |
| 5.1.4.  | CommunicationMatrixInstance (from SystemTemplate).....                                   | 65 |
| 5.1.5.  | SystemMapping (from SystemTemplate) .....  | 66 |
| 5.1.6.  | SoftwareComposition (from SystemTemplate).....   | 66 |
| 5.2.    | SystemSignal .....   | 67 |
| 5.2.1.  | SystemSignal (from SystemTemplate::SystemSignal) .....                                   | 67 |
| 5.2.2.  | ClusterSignal (from SystemTemplate::SystemSignal).....                                   | 68 |
| 5.2.3.  | ClusterSignalReceiver (from SystemTemplate::SystemSignal).....                           | 68 |
| 5.2.4.  | ClusterSignalTransmitter (from SystemTemplate::SystemSignal).....                        | 68 |
| 5.3.    | FrameStructure .....   | 69 |
| 5.3.1.  | FrameType (from SystemTemplate::FrameStructure).....                                     | 69 |
| 5.3.2.  | SignalMultiplexer (from SystemTemplate::FrameStructure).....                             | 70 |
| 5.3.3.  | SubFrame (from SystemTemplate::FrameStructure) .....                                     | 70 |
| 5.3.4.  | SignalPosition (from SystemTemplate::FrameStructure) .....                               | 71 |
| 5.3.5.  | SignalOutdatedIndication (from SystemTemplate::FrameStructure) ...                       | 71 |
| 5.3.6.  | SignalUpdateIndication (from SystemTemplate::FrameStructure) .....                       | 71 |
| 5.4.    | DataMapping.....   | 72 |
| 5.4.1.  | DataMapping (from SystemTemplate::DataMapping) .....                                     | 72 |
| 5.4.2.  | SenderReceiverToUnspecifiedConnectionMapping (from<br>SystemTemplate::DataMapping) ..... | 72 |
| 5.4.3.  | SenderReceiverToSignalMapping (from<br>SystemTemplate::DataMapping) .....                | 72 |
| 5.5.    | SystemMappingReference .....   | 73 |
| 5.5.1.  | SwCompToEcuMapping (from SystemTemplate::SWmapping).....                                 | 73 |
| 5.6.    | Communication .....  | 74 |
| 5.6.1.  | CommunicationMatrixType (from SystemTemplate::Communication) .....                       | 74 |
| 5.6.2.  | LinSchedulingTable (from SystemTemplate::Communication).....                             | 74 |
| 5.6.3.  | LinSchedulingTableEntry (from SystemTemplate::Communication) ...                         | 75 |
| 5.6.4.  | EventTriggeredFrameSlot (from SystemTemplate::Communication) ..                          | 75 |
| 5.6.5.  | UnconditionalFrameSlot (from SystemTemplate::Communication) .....                        | 75 |
| 5.6.6.  | SporadicFrameSlot (from SystemTemplate::Communication) .....                             | 76 |
| 5.6.7.  | FrameInstance (from SystemTemplate::Communication) .....                                 | 76 |
| 5.6.8.  | LinFrameInstance (from SystemTemplate::Communication) .....                              | 76 |
| 5.6.9.  | CanFrameInstance (from SystemTemplate::Communication).....                               | 77 |
| 5.6.10. | FlexrayFrameInstance (from SystemTemplate::Communication)....                            | 77 |
| 5.6.11. | SystemEventTrigger (from SystemTemplate::Communication) .....                            | 77 |
| 5.6.12. | SystemStateTrigger (from SystemTemplate::Communication).....                             | 78 |
| 5.6.13. | ActiveCondition (from SystemTemplate::Communication) .....                               | 78 |
| 5.6.14. | SendCondition (from SystemTemplate::Communication) .....                                 | 79 |
| 5.6.15. | StopCondition (from SystemTemplate::Communication) .....                                 | 79 |
| 5.6.16. | StartCondition (from SystemTemplate::Communication).....                                 | 79 |
| 5.6.17. | SignalTrigger (from SystemTemplate::Communication) .....                                 | 80 |
| 5.6.18. | Timing (from SystemTemplate::Communication).....   | 80 |
| 5.6.19. | UnspecifiedTiming (from SystemTemplate::Communication) .....                             | 80 |
| 5.6.20. | EventControlledTiming (from SystemTemplate::Communication) ...                           | 81 |
| 5.6.21. | CyclicTiming (from SystemTemplate::Communication) .....                                  | 81 |
| 5.7.    | Topology .....   | 82 |
| 5.7.1.  | SystemTopologyType (from SystemTemplate::Topology) .....                                 | 83 |
| 5.7.2.  | PhysicalChannel (from SystemTemplate::Topology) .....                                    | 83 |

|         |  |    |
|---------|--|----|
| 5.7.3.  | PhysicalMediumSegment (from SystemTemplate::Topology).....               | 83 |
| 5.7.4.  | CommunicationCluster (from SystemTemplate::Topology).....                | 84 |
| 5.7.5.  | MOSTBus (from SystemTemplate::Topology).....                             | 84 |
| 5.7.6.  | FlexrayCluster (from SystemTemplate::Topology).....                      | 84 |
| 5.7.7.  | LINSubBus (from SystemTemplate::Topology).....                           | 85 |
| 5.7.8.  | UnspecifiedConnection (from SystemTemplate::Topology).....               | 85 |
| 5.7.9.  | CANBus (from SystemTemplate::Topology).....                              | 86 |
| 5.7.10. | ECUInstance (from SystemTemplate: Topology).....                         | 86 |
| 5.7.11. | ECUCommunicationPortInstance (from SystemTemplate::Topology) .<br>.....  | 87 |
| 5.7.12. | FlexrayCommunicationPortInstance (from<br>SystemTemplate::Topology)..... | 87 |
| 5.7.13. | CANCommunicationPortInstance (from SystemTemplate::Topology) .<br>.....  | 88 |
| 5.7.14. | LINCommunicationPortInstance (from SystemTemplate::Topology)             | 89 |
| 5.7.15. | HUB (from SystemTemplate::Topology).....                                 | 89 |
| 5.7.16. | Bus.....   | 90 |
| 6.      | References.....  | 91 |
| 6.1.    | UML2 Data Types.....   | 91 |
| 6.2.    | Normative References to AUTOSAR documents.....                           | 91 |
| 6.3.    | Normative References to external documents.....                          | 92 |
| 6.4.    | Reference to Models.....   | 93 |

## 1. Introduction

AUTOSAR has defined a metamodel to describe the System, Software and Hardware of an automobile. Although his metamodel is described using UML it is not straightforward to describe instances of this in a UML tool.

The AUTOSAR metamodel has three<sup>1</sup> major packages for Software, System and ECU; this document reflects this structure with separate sections for each of these models.

### 1.1. Goals

It's intended that this document should provide a precise and pragmatic mapping between AUTOSAR 2.0 and UML 2.0. There are many areas of good overlap between the metamodels, however there are other areas where there is little overlap, in these circumstances some liberties may have been taken.

An important goal of this document is to minimize UML tool customization, although this cannot be avoided for a first class AUTOSAR modeling environment.

### 1.2. Scope

This document will initially only provide a mapping to a subset of the metamodel , the ECU, SW-C and System templates. It is aligned with AUTOSAR 2.0, see [4][5][6] and UML 2.0, see 92[14]

AUTOSAR has defined a graphical notation see ref [10]; when this notation affects a stereotype it will be documented.

### 1.3. Terminology

Stereotypes defined in the profile may extend any class of the UML2 metamodel – we assume the highest compliance level – level 3. In some cases, compromises have to be made in order to implement the profile in a typical UML tool-set. These cases will be clearly marked in the document.

Constraints are expressed using OCL, it is assumed that the *oclIsTypeOf()*, *oclIsKindOf()* functions are stereotype aware.

Abstract AUTOSAR classes are represented in the profile only in the following cases:

- the stereotypes that are used to represent the abstract classes contribute (constraints, attributes) to the stereotypes that specialize them
- the stereotypes that are used to represent the abstract classes allow defining constraints of other stereotypes in a more general way

---

<sup>1</sup> There is a fourth section for ECU configuration data.

Abstract AUTOSAR classes are represented by abstract stereotypes, while concrete classes are represented by concrete stereotypes.

Just like a class, a stereotype may have properties, which may be referred to as tag definitions. When a stereotype is applied to a model element, the values of the properties may be referred to as tagged values. In this document, the terms *tag definitions* and *stereotype's properties* are used interchangeably.

A composite aggregation between AUTOSAR elements will be typically modeled using the Element::ownedElement association. An association will be modeled using a stereotype attribute of type string which will contain the qualified name of the associated element in the form (/model/package/./package/Type.Property.Property); some stereotypes may be able to reuse existing UML 2.0 metamodel associations, this will be documented.

Images within this document are from three sources,

- Enterprise Architect for AUTOSAR model diagrams.
- OMG UML 2.0 Superstructure Specification, see [14] for UML 2.0 model diagrams.
- Rational Software Architect for AUTOSAR Profile diagrams, see [18],[19].

## 2. Example use of the Profile

The description of the mapping between UML and AUTOSAR is somewhat academic. This chapter attempts to show how the profile (mapping) can be used for a simple example.

These screenshots are taken from a partial wiper system (WP10.1) model built using Rational Software Modeler (RSM) with the AUTOSAR profile applied. The profile has been created using RSM's profile editor with OCL constraints and graphics also defined.

The model ([19]) and profile ([18]) can be found on the subversion server. RSM can be downloaded for free, see [20].

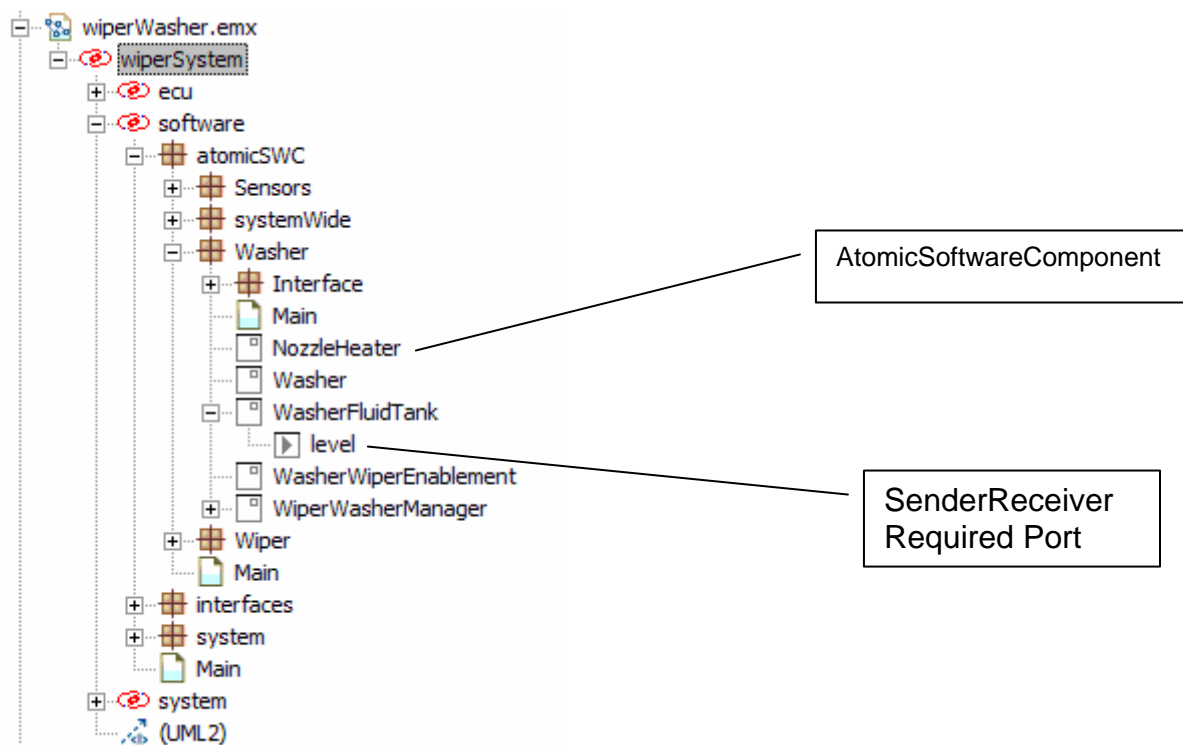


Figure 1 Model Explorer showing expansion of AUTOSAR elements

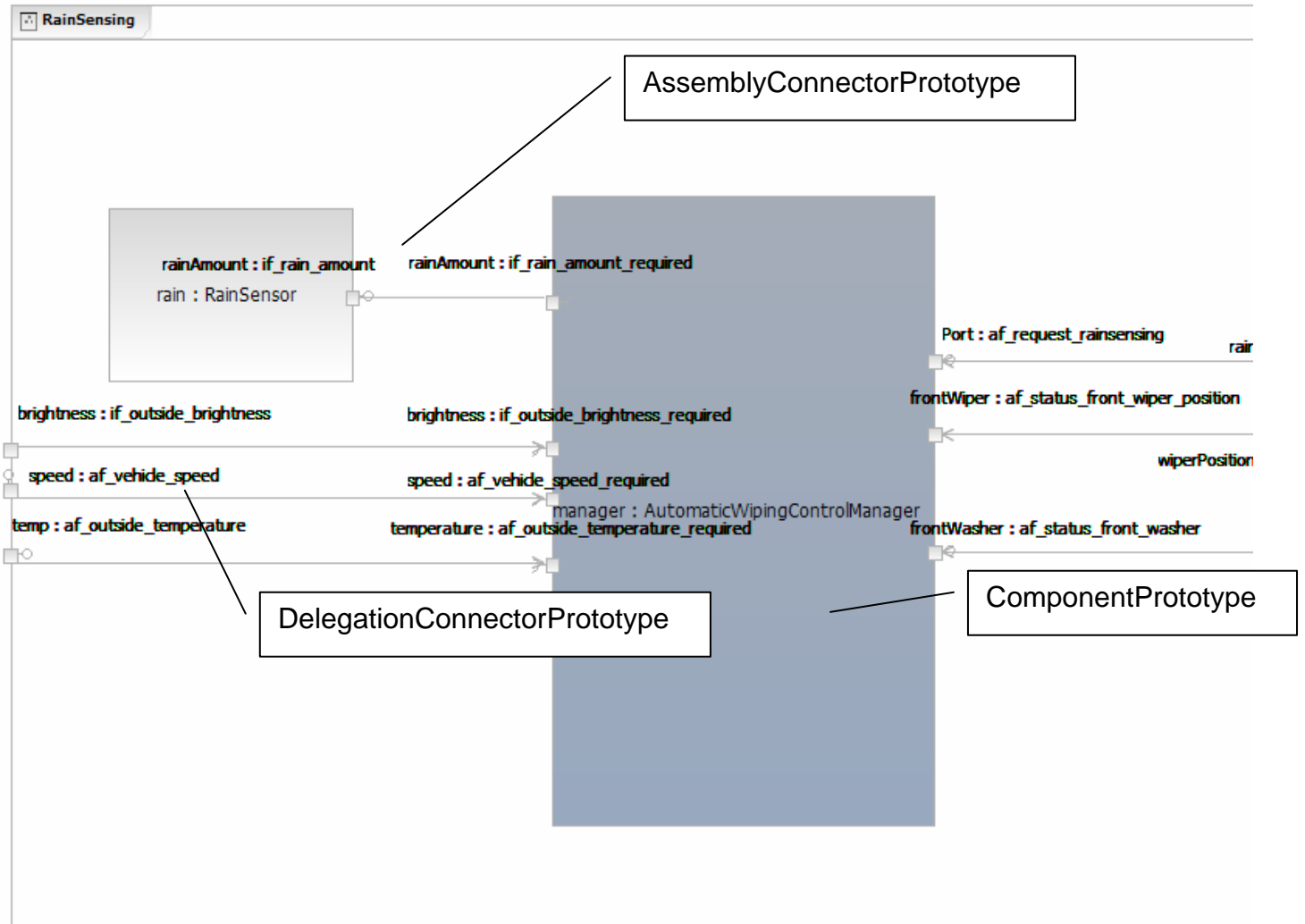


Figure 2 CompositionType internal structure of RainSensing component

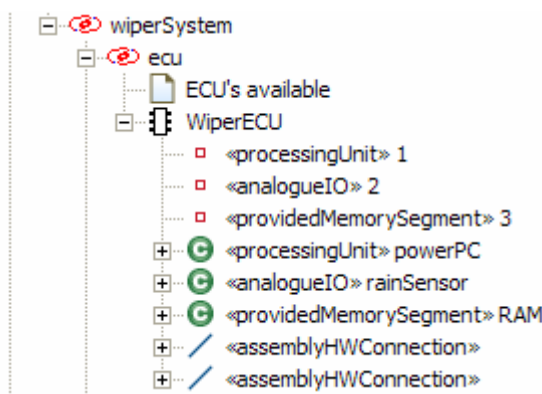
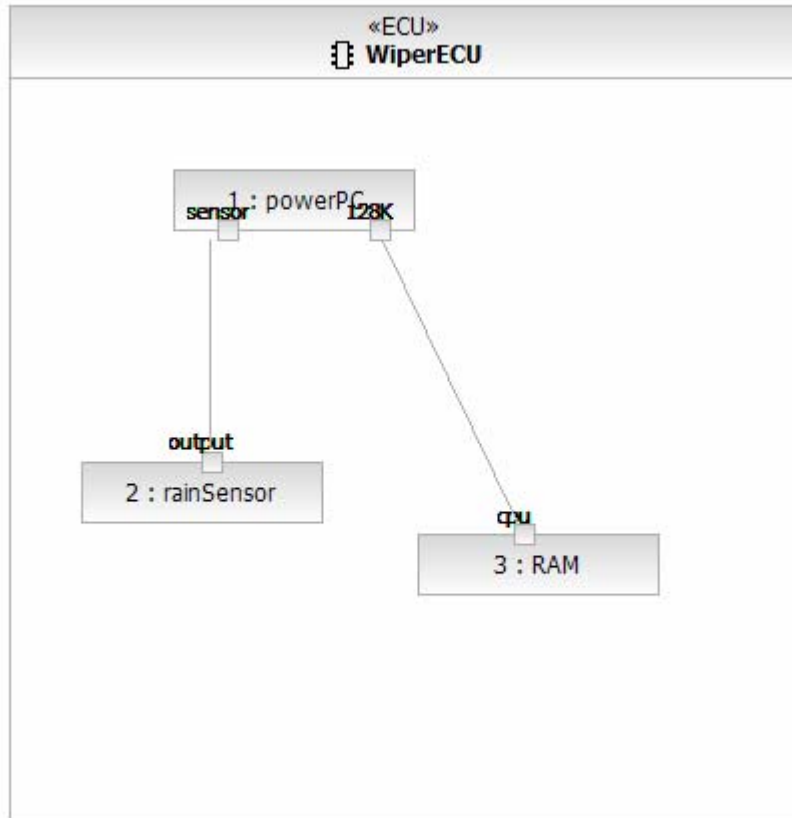
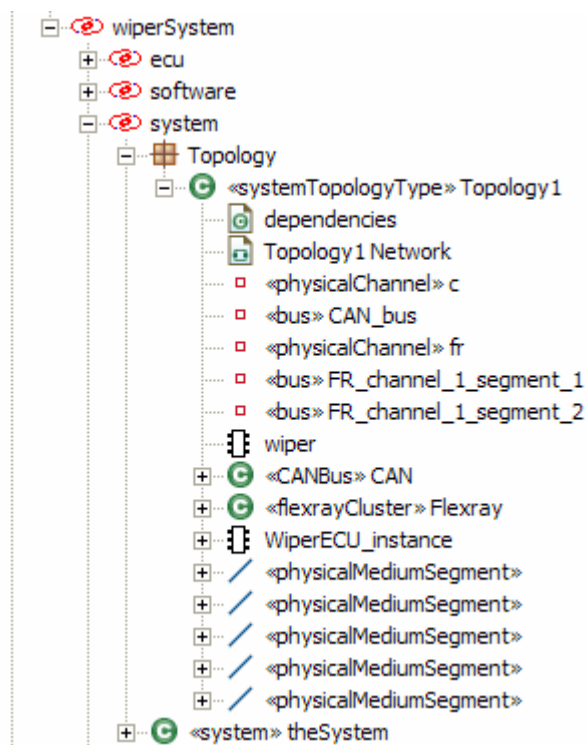


Figure 3 ECU in model explorer. Note stereotypes on both the Classes and Properties (Parts), this is a necessary workaround for UML.



**Figure 4 ECU internal structure**



**Figure 5 Model explorer showing System Topology.**

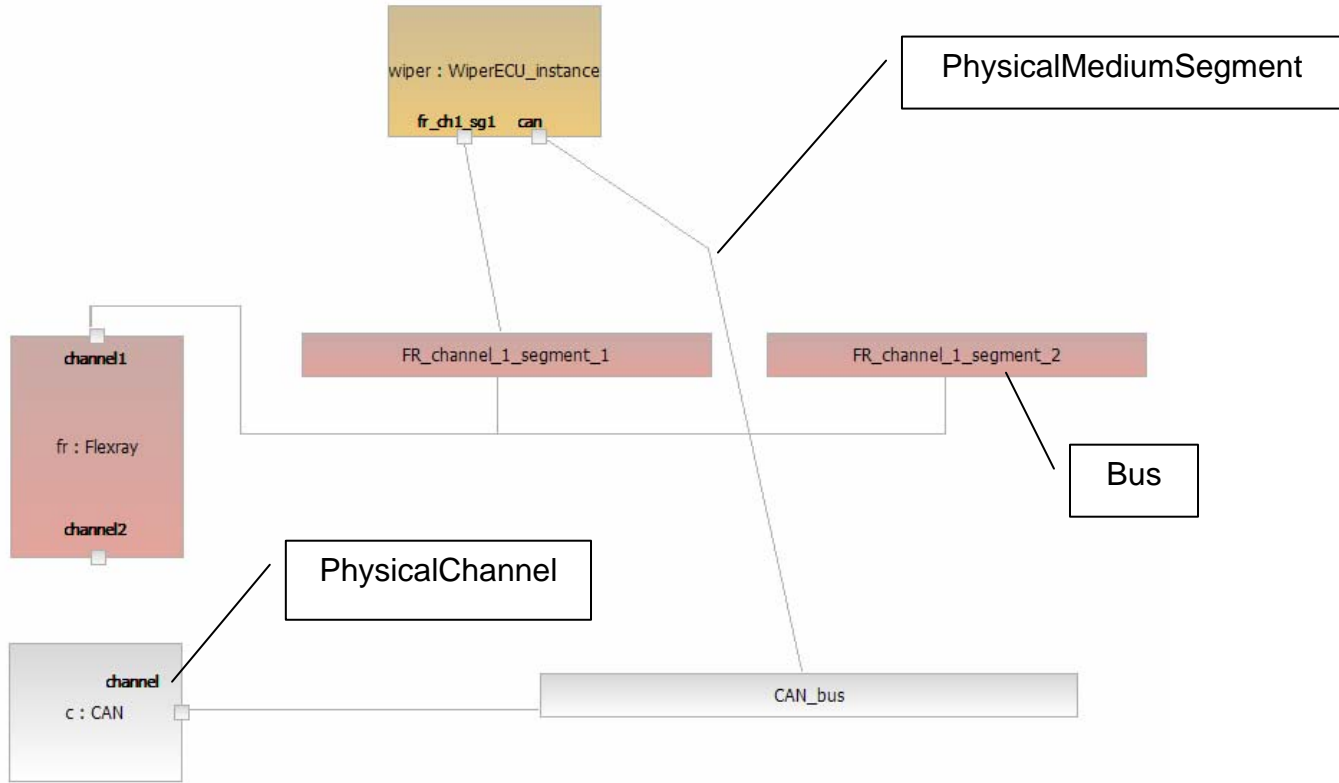


Figure 6 Topology diagram.

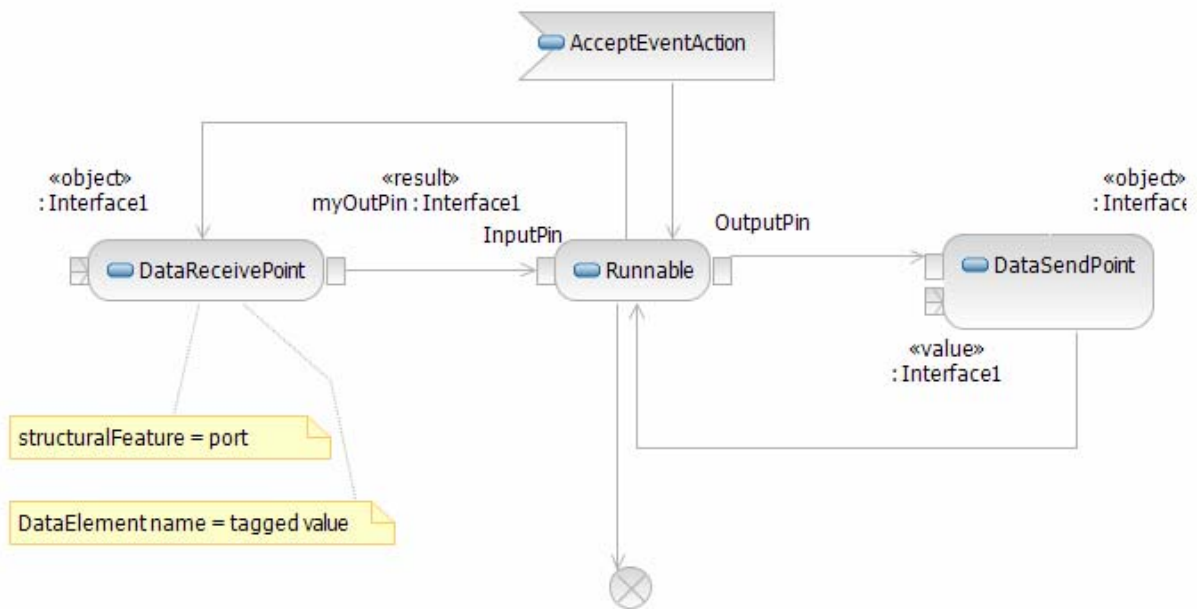


Figure 7 DataSendPoint and DataReceivePoint on an Activity Diagram

### 3. Software Component Model

This section describes the profile for the SW-C AUTOSAR model.

#### 3.1. Compositions

The purpose of an AUTOSAR composition is to allow encapsulation of functionality by aggregating existing software components. Since a composition is also a kind of component, it again may be aggregated in even further compositions. This recursive relation is shown in the following extract from the AUTOSAR metamodel.

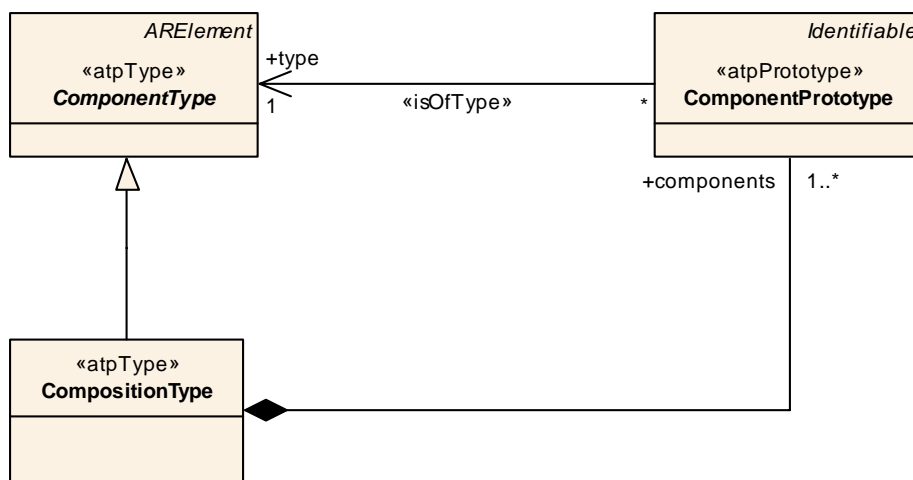


Figure 8: The recursive relation of software components and compositions.

##### 3.1.1. ComponentType (from Components)

ComponentType is an abstract class that is represented by the abstract <<componentType>> stereotype which extends the Class metaclass (from StructuredClasses).

In UML2, the Class metaclass is a kind of classifier that can own an internal structure and ports (see Figure 9). Thus, it has the closest semantic match to the ComponentType class and is chosen to represent it.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>   | <b>Constrained feature</b>              | <b>OCL Expression</b>                                  |
|---|---|---|--|
| 1 | All owned ports MUST be elements stereotyped by the <<portPrototype>> stereotype or its sub-stereotypes | EncapsulatedClassifier :<br>: ownedPort | self.ownedPort-><br>forAll(oclIsTypeOf(portPrototype)) |
| 2 | No Operations   | ownedOperation                          | self.ownedOperation->size() = 0                        |
| 3 | No Nested Classifier  | nestedClassifier                        | self.nestedClassifier->size() = 0                      |



### 3.1.3. CompositionType (from Composition)

CompositionType is represented by the <<compositionType>> stereotype which is a specialization of the <<componentType>> stereotype defined in Section 3.1.1. CompositionType is a concrete class and thus it is represented by a concrete stereotype.

CompositionType elements can own connectors and sub-components. Owned connectors are represented by the UML2 ownedConnector metaassociation, which, according to UML2 spec, references the connector owned by the classifier. Owned sub-components are represented by the UML2 part metaassociation, which references the properties specifying instances that the classifier owns by composition. See Figure 9 for more details.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>   | <b>Constrained feature</b>           | <b>OCL Expression</b>   |
|---|---|--------------------------------------|---|
| 1 | All owned connectors MUST be elements stereotyped by the <<connectorPrototype>> stereotype or its sub-stereotypes     | StructuredClassifier::ownedConnector | self.ownedConnector->forAll(oclIsKindOf(connect orPrototype)) |
| 2 | All owned sub-components MUST be elements stereotyped by the <<componentPrototype>> stereotype or its sub-stereotypes | StructuredClassifier::part           | self.part->forAll(oclIsKindOf(componentPrototype))            |

For Notation, see [10] 3.3.2.1.

### 3.1.4. AtomicSoftwareComponentType (from Components)

AtomicSoftwareComponentType is represented by the <<atomicSoftwareComponentType>> stereotype which is a specialization of the <<componentType>> stereotype defined in 3.1.1. AtomicSoftwareComponentType is a concrete class and thus it is represented by a concrete stereotype.

AtomicSoftwareComponentType does not allow encapsulation of functionality by aggregating existing software components as well as connections between them.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>     | <b>Constrained feature</b>           | <b>OCL Expression</b>         |
|---|-----------------------------------|--------------------------------------|-------------------------------|
| 1 | MUST NOT contain owned connectors | StructuredClassifier::ownedConnector | self.ownedConnector->size()=0 |
| 2 | MUST NOT contain owned parts      | StructuredClassifier::part           | self.part->size()=0           |

For Notation, see [10] 3.3.2.1.

### 3.1.5. SensorActuatorSoftwareComponentType (from Components)

SensorActuatorSoftwareComponentType is represented by the <<sensorActuatorSoftwareComponentType>> stereotype which is a specialization of the <<atomicSoftwareComponentType>> stereotype defined in 3.1.4. SensorActuatorSoftwareComponentType is a concrete class and thus it is represented by a concrete stereotype.

This class has a relationship to a SensorActuatorHW class, this will be modeled using a string attribute which will contain the qualified name of the SensorActuatorHW.

No additional semantic constraints are defined for this stereotype.

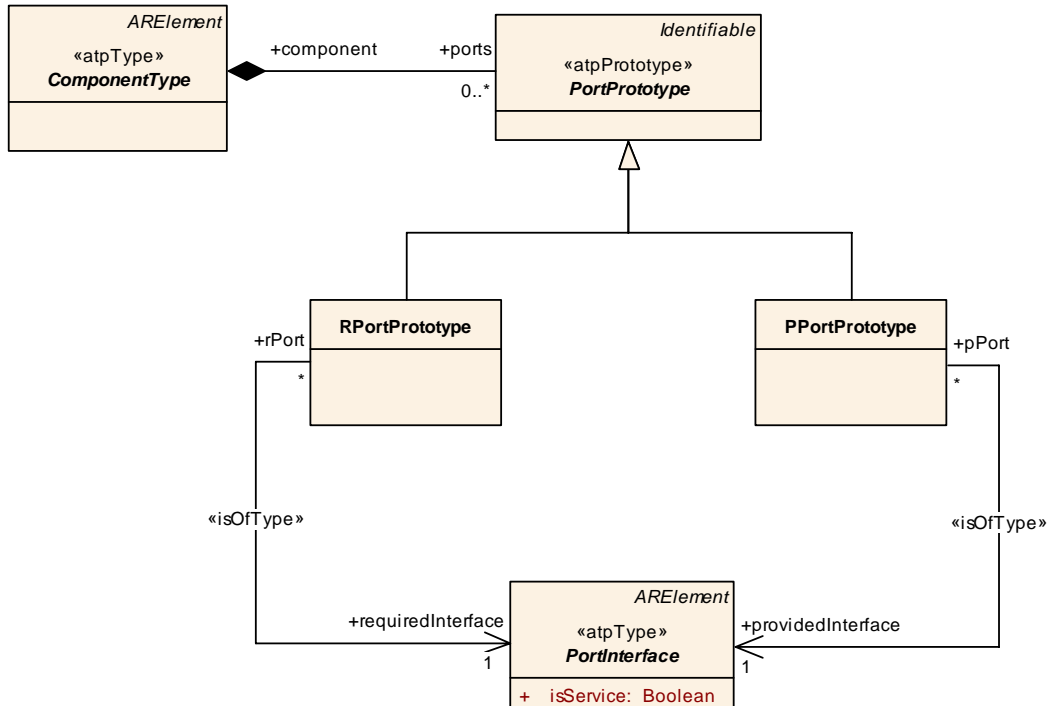
The following attributes are required for this stereotype

|   | <b>Attribute Description</b>  | <b>Name</b>    | <b>Type</b> | <b>Multiplicity</b> |
|---|---|----------------|-------------|---------------------|
| 1 | SensorActuatorHW reference represented by a String. String format will be (/model/package/./package/Type.Property.Property) | sensorActuator | String      | 1                   |

For Notation, see [10] 3.3.2.1.

### 3.2. Ports and Connectors

AUTOSAR software components have well defined interaction points to describe the possible kinds of (data as well as service oriented) communication with other software components, called the ports of the component. Figure 10: AUTOSAR Software Components, Ports and Interface shows the fundamental classes from the metamodel.



**Figure 10: AUTOSAR Software Components, Ports and Interface**

Compositions contain two kinds of connectors: assembly connectors to interconnect ports of components that are part of the composition as well as delegation connectors from inner ports to delegated ports. This is shown in the following diagram.

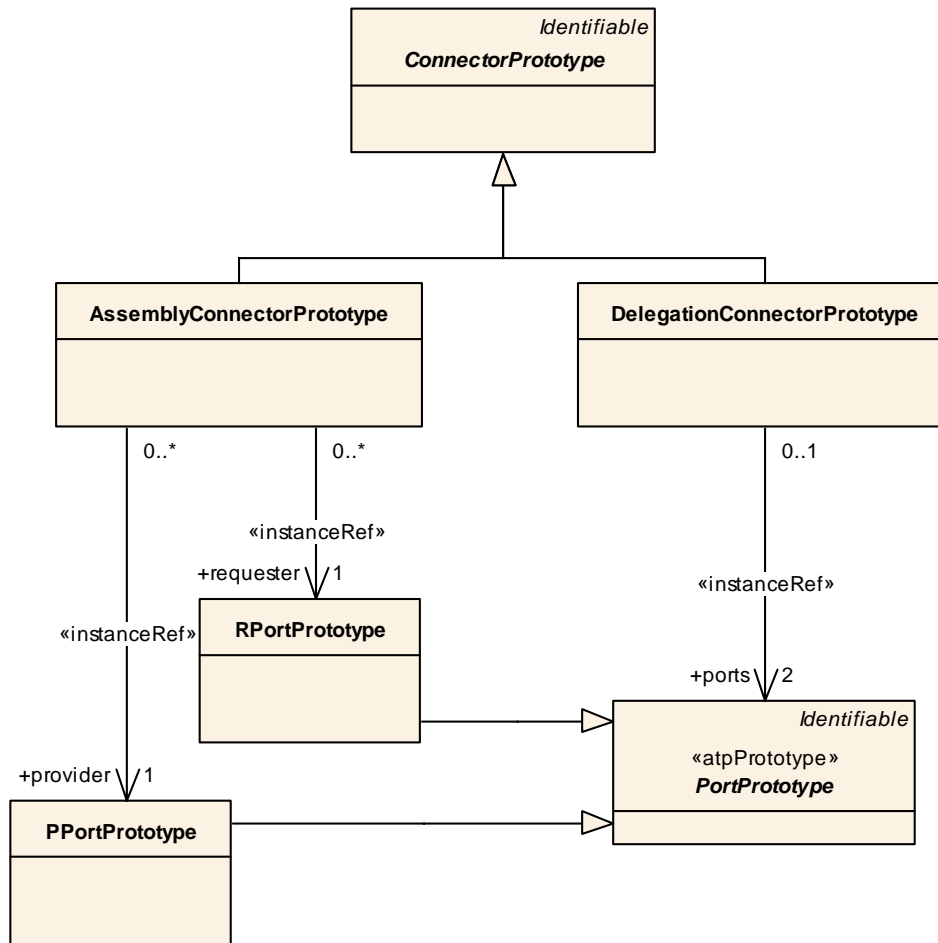


Figure 11: Connector types and the ports they reference.

### 3.2.1. PortPrototype (from Components)

PortPrototype is an abstract class that is represented by the abstract <<portPrototype>> stereotype which extends the Port metaclass (from Ports).

The Port metaclass is chosen because of its closest semantic match to the Port class.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>                                 | <b>Constrained feature</b> | <b>OCL Expression</b>  |
|---|---|----------------------------|--|
| 1 | Service port for intra-layer communication                    | Port::isService            | self.isService=true  |
| 2 | If owned by an AtomicSoftwareComponent, must be behavior port | Port::isBehavior           | self.owner.oclsTypeOf(AtomicSoftwareComponent)<br>implies self.isBehavior = true |

### 3.2.2. PPortPrototype (from Components)

PPortPrototype is represented by the <<pPortPrototype>> stereotype which is a specialization of the <<portPrototype>> stereotype defined in 3.2.1. PPortPrototype is a concrete class and thus it is represented by a concrete stereotype.

In UML2, the required and provided interfaces on ports are derived collections (see Figure 9). A port can be typed by a class or an interface. If a port is typed by a class then it provides all the interfaces that the class realizes. If a port is typed by an interface, it provides that interface. In AUTOSAR, PPortPrototype references exactly one provided interface (element stereotyped by the <<portInterface>> stereotype), thus, the port representing PPortPrototype is typed by an interface representing the provided AUTOSAR interface.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>  | <b>Constrained feature</b> | <b>OCL Expression</b>                    |
|---|--|----------------------------|--|
| 1 | The port MUST only be typed by a UML2 Interface which has the <<portInterface>> stereotype (or subclass) | TypedElement::type         | self.provided.ocllsKindOf(portInterface) |
| 2 | The port MUST provide exactly one interface  | Port:: provided            | provided->size() = 1                     |
| 3 | The port MUST NOT require any interfaces   | Port::required             | required->size() = 0                     |

For Notation, see [10] 3.3.3.1. Note that the decoration within the port is defined by the type of the interface, SenderReceiver or ClientServer, for the former the decoration must be rotated to always point outwards.

### 3.2.3. ServerPort

ServerPort is represented by the <<serverPort>> stereotype which is a specialization of the <<pPortPrototype>> stereotype defined in Section 3.2.2. ServerPort is a concrete class and thus it is represented by a concrete stereotype.

For compatibility with generic UML tools PPortPrototype has been further refined so that a ProvidedPort typed with a ClientServer interface has its own stereotype and therefore can be directly created.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>   | <b>Constrained feature</b> | <b>OCL Expression</b>                            |
|---|---|----------------------------|--|
| 1 | The port MUST be typed by a UML2 Interface which has the <<ClientServerInterface>> stereotype | TypedElement::type         | self.provided.ocllsTypeOf(ClientServerInterface) |

For Notation, see [10] 3.3.3.1.

### 3.2.4. ReceiverPort

ReceiverPort is represented by the <<receiverPort>> stereotype which is a specialization of the <<rPortPrototype>> stereotype defined in 3.2.5. ReceiverPort is a concrete class and thus it is represented by a concrete stereotype.

For compatibility with generic UML tools RPortPrototype has been further refined so that a ProvidedPort typed with a SenderReceiver interface has its own stereotype and therefore can be directly created.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>   | <b>Constrained feature</b> | <b>OCL Expression</b>                              |
|---|---|----------------------------|--|
| 1 | The port MUST be typed by the UML2 Interface which has the <<SenderReceiverInterface>> stereotype | TypedElement::type         | self.required.ocllsTypeOf(SenderReceiverInterface) |

For Notation, see [10] 3.3.3.1.

### 3.2.5. RPortPrototype (from Components)

RPortPrototype is represented by the <<rPortPrototype>> stereotype which is a specialization of the <<portPrototype>> stereotype defined in Section 3.2.1. RPortPrototype is a concrete class and thus it is represented by a concrete stereotype.

In UML2, the required and provided interfaces on Ports are derived collections see Figure 9). A port can be typed by a class or an interface. If a port is typed by a class then it requires all the interfaces that the class requires. The act of typing a port directly with an interface is a shortcut for ports that only provide 1 interface. In AUTOSAR, RPortPrototype references exactly one required interface (element stereotyped by the <<portInterface>> stereotype), thus, the port representing RPortPrototype is typed by a class that requires exactly one interface.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>  | <b>Constrained feature</b> | <b>OCL Expression</b>  |
|---|--|----------------------------|--|
| 1 | The port MUST be typed by the UML2 Class metaclass which must have a Usage relationship to exactly one <<portInterface>> | TypedElement::type         | self.type.ocllsTypeOf(Class) implies self.type.clientDependency->forAll( d   d.ocllsTypeOf(Usage) implies d.supplier.ocllsTypeOf(portInterface)) |
| 2 | The port MUST require exactly one interface  | Port::required             | required->size() = 1   |
| 3 | The port MUST NOT provide any interfaces   | Port::provided             | provided->size() = 0   |

For Notation, see [10] 3.3.3.1. Note that the decoration within the port is defined by the type of the interface, SenderReceiver or ClientServer, for the former the decoration must be rotated to always point inwards.

### 3.2.6. ClientPort

ClientPort is represented by the <<clientPort>> stereotype which is a specialization of the <<rPortPrototype>> stereotype defined in Section 3.2.5. ClientPort is a concrete class and thus it is represented by a concrete stereotype.

For compatibility with generic UML tools RPortPrototype has been further refined so that a RequiredPort typed with a ClientServer has its own stereotype and therefore can be directly created.

There are no semantic constraints defined.

For Notation, see [10] 3.3.3.1.

### 3.2.7. SenderPort

SenderPort is represented by the <<senderPort>> stereotype which is a specialization of the <<pPortPrototype>> stereotype defined in Section 3.2.2. SenderPort is a concrete class and thus it is represented by a concrete stereotype.

For compatibility with generic UML tools PPortPrototype has been further refined so that a RequiredPort typed with a SenderReceiver interface has its own stereotype and therefore can be directly created.

There are no semantic constraints defined.

For Notation, see [10] 3.3.3.1.

### 3.2.8. ConnectorPrototype (from Composition)

ConnectorPrototype is an abstract class that is represented by the abstract <<connectorPrototype>> stereotype which extends the Connector metaclass (from InternalStructures).

In UML2, Connector may be attached to two or more connectable elements, each representing a set of instances (see Figure 12). Since port is a connectable element (see Figure 9), UML2's Connector can be used to connect port instances and thus is chosen to represent the ConnectorPrototype.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>                               | <b>Constrained feature</b> | <b>OCL Expression</b> |
|---|---|----------------------------|-----------------------|
| 1 | Connector prototype MUST connect exactly two connector ends | Connector::end             | self.end->size() = 2  |
| 2 | The connector MUST NOT be typed by                          | Connector::type            | self.type->size() = 0 |

|                 |  |  |
|-----------------|--|--|
| any association |  |  |
|-----------------|--|--|

For Notation, see [10] 3.3.6.1.

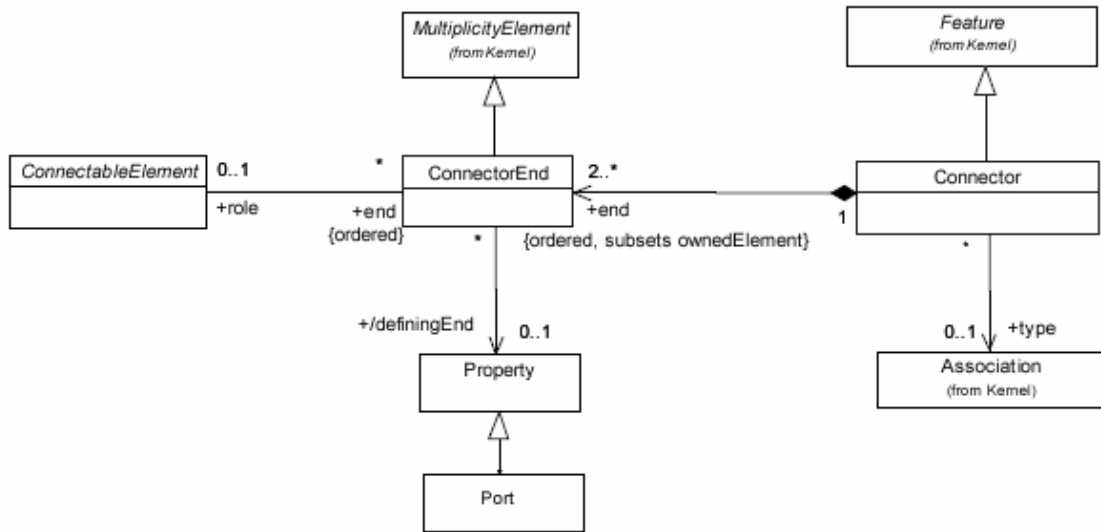


Figure 12: UML2 Connectors

### 3.2.9. AssemblyConnectorPrototype (from Composition)

AssemblyConnectorPrototype is represented by the <<assemblyConnectorPrototype>> stereotype which is a specialization of the <<connectorPrototype>> stereotype defined in Section 3.2.8. AssemblyConnectorPrototype is a concrete class and thus it is represented by a concrete stereotype.

Assembly connectors are used to interconnect a required and a provided port of components that are part of a composition.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>  | <b>Constrained feature</b> | <b>OCL Expression</b>                                  |
|---|--|----------------------------|--|
| 1 | There MUST be a connector end which role association references an element stereotyped by the <<pPortPrototype>> | ConnectorEnd::role         | self.end->one( e   e.role.ocllsTypeOf(pPortPrototype)) |
| 2 | There MUST be a connector end which role association references an element stereotyped by the <<rPortPrototype>> | ConnectorEnd::role         | self.end->one( e   e.role.ocllsTypeOf(rPortPrototype)) |
| 3 | Connector ends must be compatible (see SWC Document for interface compatibility)                                 | ConnectorEnd::role         | not defined.   |

For Notation, see [10] 3.3.6.1.

### 3.2.10. DelegationConnectorPrototype (from Composition)

DelegationConnectorPrototype is represented by the <<delegationConnectorPrototype>> stereotype which is a specialization of the <<connectorPrototype>> stereotype defined in Section 3.2.8. DelegationConnectorPrototype is a concrete class and thus it is represented by a concrete stereotype.

Delegation connectors are used to delegate connectors from inner ports to delegated ports.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>   | <b>Constrained feature</b> | <b>OCL Expression</b> |
|---|---|----------------------------|-----------------------|
| 1 | Role association of both connector ends MUST reference elements stereotyped by the same stereotype – either <<pPortPrototype>> or <rPortPrototype>> | ConnectorEnd::role         | not defined           |

For Notation, see [10] 3.3.6.1.

### 3.3. Interfaces

A Port interface defines which information is exchanged between software components, or to be more precise between ports of those components. It does this by formally describing the names and signatures of operations and data elements exchanged between the two components.

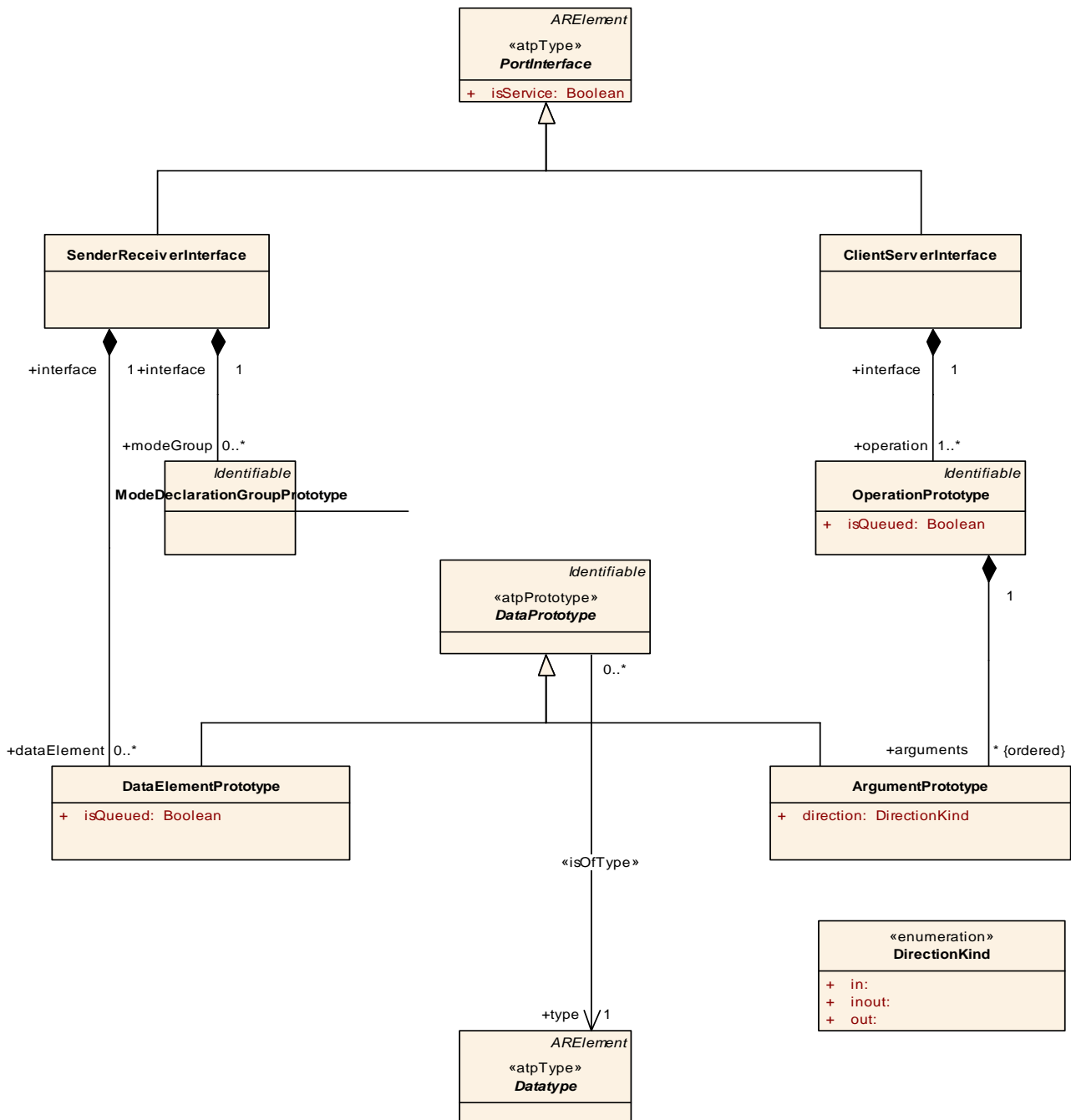


Figure 13: Interfaces in the AUTOSAR metamodel.

Sender/receiver interfaces allow for the specification of the common, typically asynchronous, communication pattern where a sender provides data that is required

by one or more receivers. While the actual communication takes place via the respective ports, s/r interfaces allow a formal description of what kind of data is sent and received. Interfaces currently are limited to describe the static structure of this exchanged information, while the communication relevant dynamic attributes are attached to ports.

Data elements serve as the data units that are exchanged between sender and receiver. A data element or more precisely a “DataElementPrototype” is created by choosing a data type – the datatype of the data element – and a name for the data element.

See Section 6.1 for more details about UML2 data types.

Client/server interfaces aggregate a number of operations. They are the service oriented counterpart to the sender/receiver interfaces mentioned above.

### 3.3.1. PortInterface (from PortInterface)

PortInterface is an abstract class that is represented by the abstract <<portInterface>> stereotype which extends the Interface metaclass (from Interfaces).

In UML2, an interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations. It has the closest semantic match to the PortInterface class; in particular, Interface can own properties and operations (see Figure 14). For that reason, the Interface metaclass is chosen to represent PortInterface.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>                     | <b>Constrained feature</b>  | <b>OCL Expression</b>        |
|---|---|-----------------------------|------------------------------|
| 1 | <<portInterface>> MUST NOT own nested classifiers | Interface::nestedClassifier | nestedClassifier->size() = 0 |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 | Service Port. Default = true | isService   | Boolean     | 1                   |

For Notation, see [10] 3.3.4.1.

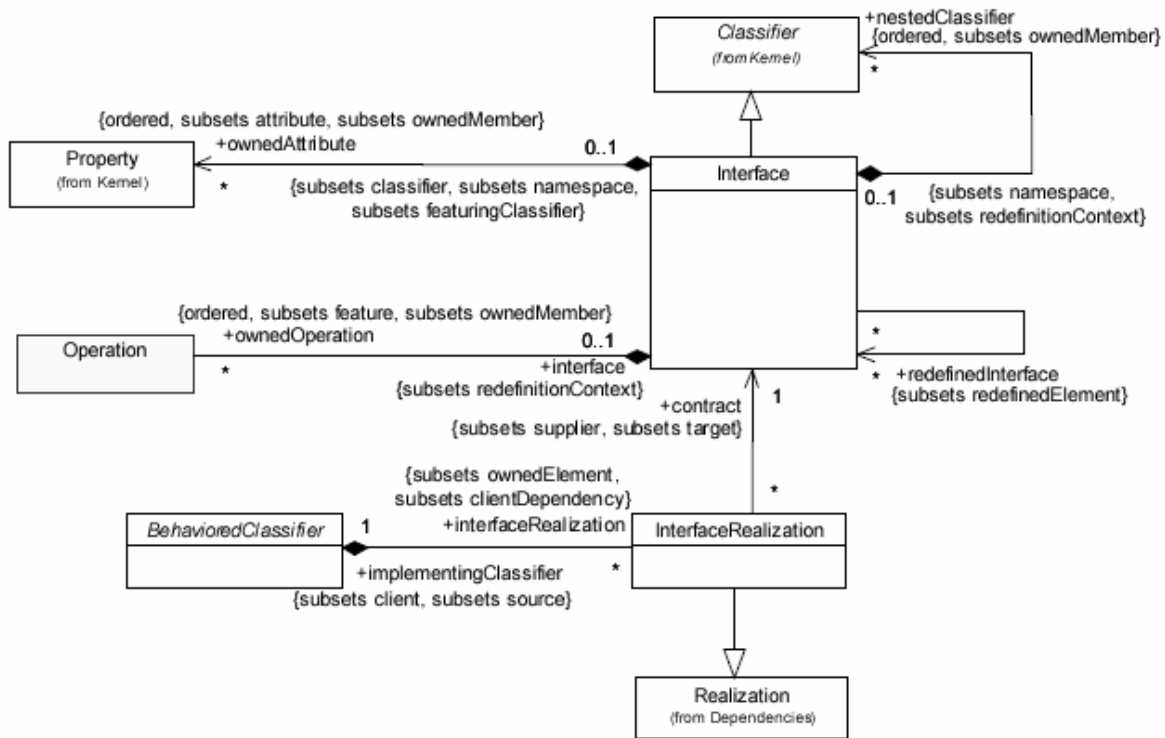


Figure 14: UML2 Interface package metaclasses

### 3.3.2. SenderReceiverInterface (from PortInterface)

SenderReceiverInterface is represented by the <<senderReceiverInterface>> stereotype which is a specialization of the <<portInterface>> stereotype defined in Section 3.3.1. SenderReceiverInterface is a concrete class and thus it is represented by a concrete stereotype.

SenderReceiverInterface interface declares a number of data elements to be sent and received. It does not declare any operations.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>  | <b>Constrained feature</b>    | <b>OCL Expression</b>  |
|---|--|-------------------------------|--|
| 1 | SenderReceiverInterface MUST NOT own operations  | Interface::<br>ownedOperation | ownedOperation ->size() = 0                                    |
| 2 | SenderReceiverInterface MUST own at least one attribute                                      | Interface::<br>ownedAttribute | ownedAttribute->size() > 0                                     |
| 3 | All owned attributes MUST BE elements stereotyped by the <<dataElementPrototype>> stereotype | Interface::<br>ownedAttribute | self.ownedAttribute->forAll(oclIsTypeOf(DataElementPrototype)) |

### 3.3.3. DataPrototype (from Datatype::Datatypes)

DataPrototype is an abstract class. There is no point in represent it in the profile. Instead, every one of its concrete subclass will be represented in the profile.

DataPrototype has four concrete uses (subclasses):

1. As a data element of an interface - DataElementPrototype: see Section 3.3.4.
2. As an argument of an operation - ArgumentPrototype (not part of the first subset, but will later be represented by the Parameter metaclass).
3. As a part of the more complex data type (struct): this case is not modeled in the AUTOSAR templates yet, and, thus, will be handled later.
4. As an interrunnable variable.

### 3.3.4. DataElementPrototype (from PortInterface)

DataElementPrototype is represented by the <<dataElementPrototype>> stereotype which extends the Property metaclass (from Kernel, AssociationClasses). DataElementPrototype is a concrete class and thus it is represented by a concrete stereotype.

In AUTOSAR, the DataElementPrototype class is used to represent data elements of a sender/receiver interface. Similarly, in UML2, Interface can own properties, which are referenced by the ownedAttribute operations (see Figure 14). Thus, it is natural to represent DataElementPrototype by the UML2 Property metaclass and to use the ownedAttribute association to reference it.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 |                              | isQueued    | Boolean     | 1                   |

### 3.3.5. ClientServerInterface (from PortInterface)

ClientServerInterface is represented by the <<clientServerInterface>> stereotype which is a specialization of the <<portInterface>> stereotype defined in Section 3.3.1. ClientServerInterface is a concrete class and thus it is represented by a concrete stereotype.

ClientServerInterface interface declares a number of operations that can be invoked on a server by a client. It does not declare any data elements.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>                    | <b>Constrained feature</b> | <b>OCL Expression</b>            |
|---|--|----------------------------|----------------------------------|
| 1 | ClientServerInterface MUST NOT own data elements | Interface::ownedAttribute  | self.ownedAttribute ->size() = 0 |

|   |  |                            |  |
|---|--|----------------------------|--|
| 2 | ClientServerInterface MUST own at least one operation                                      | Interface::owned Operation | self.ownedOperation ->size() > 0                             |
| 3 | All owned operations MUST BE elements stereotyped by the <<operationPrototype>> stereotype | Interface::owned Operation | self.ownedOperation->forAll(oclIsTypeOf(OperationPrototype)) |

### 3.3.6. OperationPrototype (from PortInterface)

OperationPrototype is represented by the <<OperationPrototype>> stereotype which extends the Operation metaclass (from Kernel, Interfaces). OperationPrototype is a concrete class and thus it is represented by a concrete stereotype.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>        | <b>Constrained feature</b> | <b>OCL Expression</b>                         |
|---|--------------------------------------|----------------------------|---|
| 1 | Owned by a <<clientServerInterface>> | Element::owner             | Self.owner.oclIsTypeOf(clientServerInterface) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 |                              | isQueued    | Boolean     | 1                   |

### 3.3.7. ArgumentPrototype (from PortInterface)

ArgumentPrototype is represented by the <<argumentPrototype>> stereotype which extends the Property metaclass (from Kernel, AssociationClasses). ArgumentPrototype is a concrete class and thus it is represented by a concrete stereotype.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>     | <b>Constrained feature</b> | <b>OCL Expression</b>                      |
|---|-----------------------------------|----------------------------|--|
| 1 | Owned by a <<operationPrototype>> | Element::owner             | Self.owner.oclIsTypeOf(operationPrototype) |

The following attributes are required for this stereotype

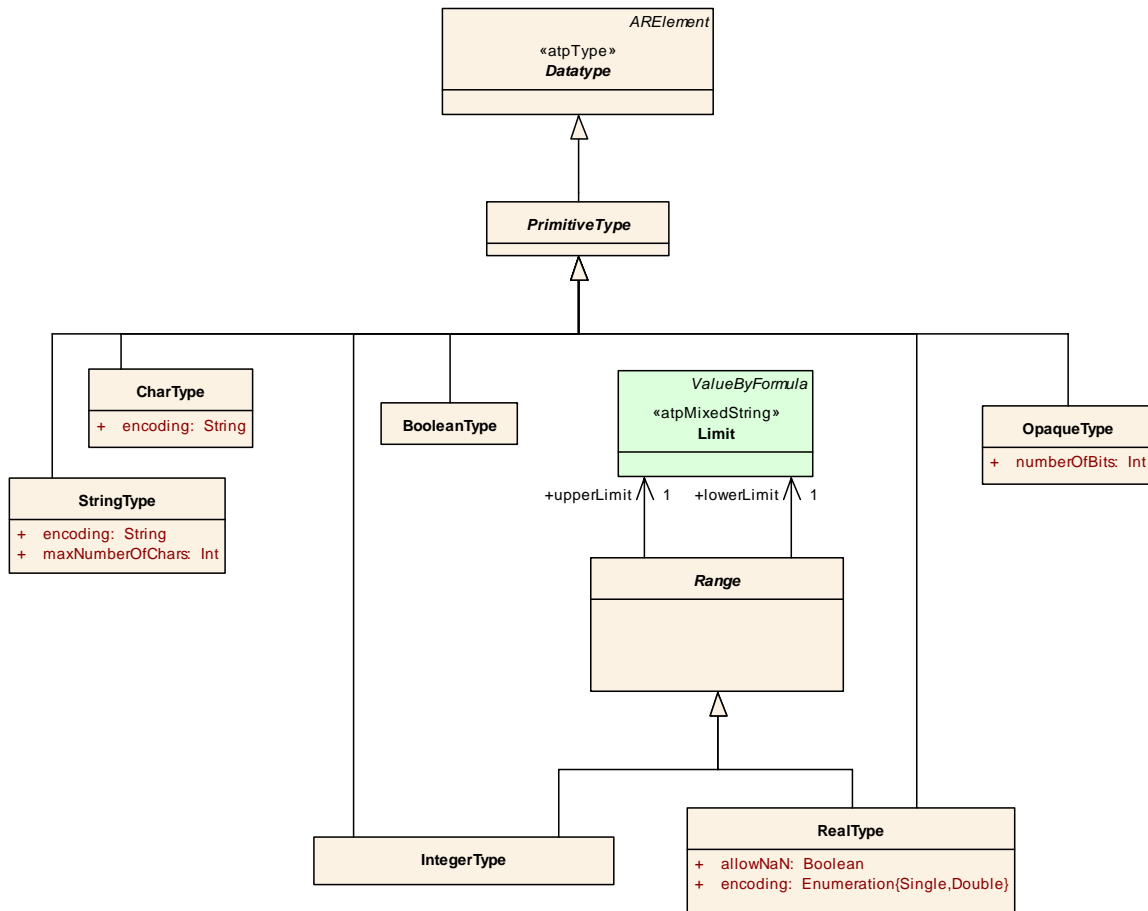
|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b>        | <b>Multiplicity</b> |
|---|------------------------------|-------------|--------------------|---------------------|
| 1 |                              | Direction   | DirectionKind_Enum | 1                   |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Type</b>        | <b>Literals</b> |
|---|--------------------------------|--------------------|-----------------|
| 1 |                                | DirectionKind_Enum | in, out, inout  |

### 3.4. Data Types

AUTOSAR provides a set of primitive data types.



**Figure 15: Primitive datatype**

Data-semantics can be attached to primitive types. It specifies how to convert between physical values (including the physical unit) and the corresponding representation of a computer system. The actual semantics class is called “SW-COMPU-METHOD”, due to compatibility with MSR-SW, see [4].

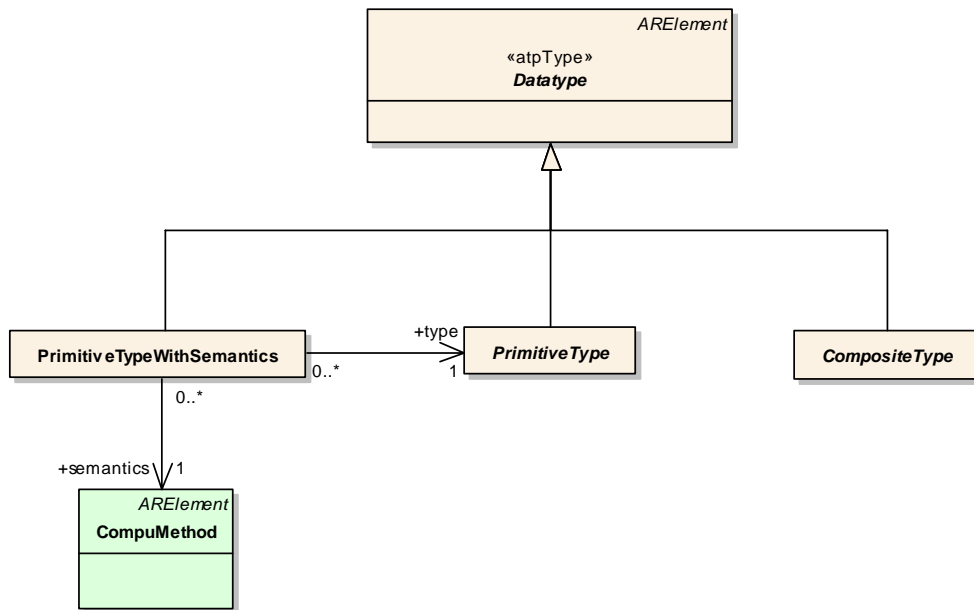


Figure 16: Primitive types and semantics.

### 3.4.1. PrimitiveType (from Datatype::Datatypes)

PrimitiveType is an abstract class and is represented by the abstract <<primitiveType>> stereotype which extends the PrimitiveType metaclass (from Kernel).

In UML2, a primitive type defines a predefined data type, without any relevant substructure (i.e. it has no parts). Primitive types may contain attributes and operations, which support the modeling of structured data types. Instances of primitive types do not have identity: if two instances have the same representation, then they are indistinguishable. In AUTOSAR, PrimitiveType has similar semantic meaning, except that it can't represent a structured data type.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>         | <b>Constrained feature</b> | <b>OCL Expression</b>            |
|---|---------------------------------------|----------------------------|----------------------------------|
| 1 | PrimitiveType MUST NOT own attributes | DataType::ownedAttribute   | self.ownedAttribute ->size() = 0 |
| 2 | PrimitiveType MUST NOT own operations | DataType::ownedOperation   | self.ownedOperation ->size() = 0 |

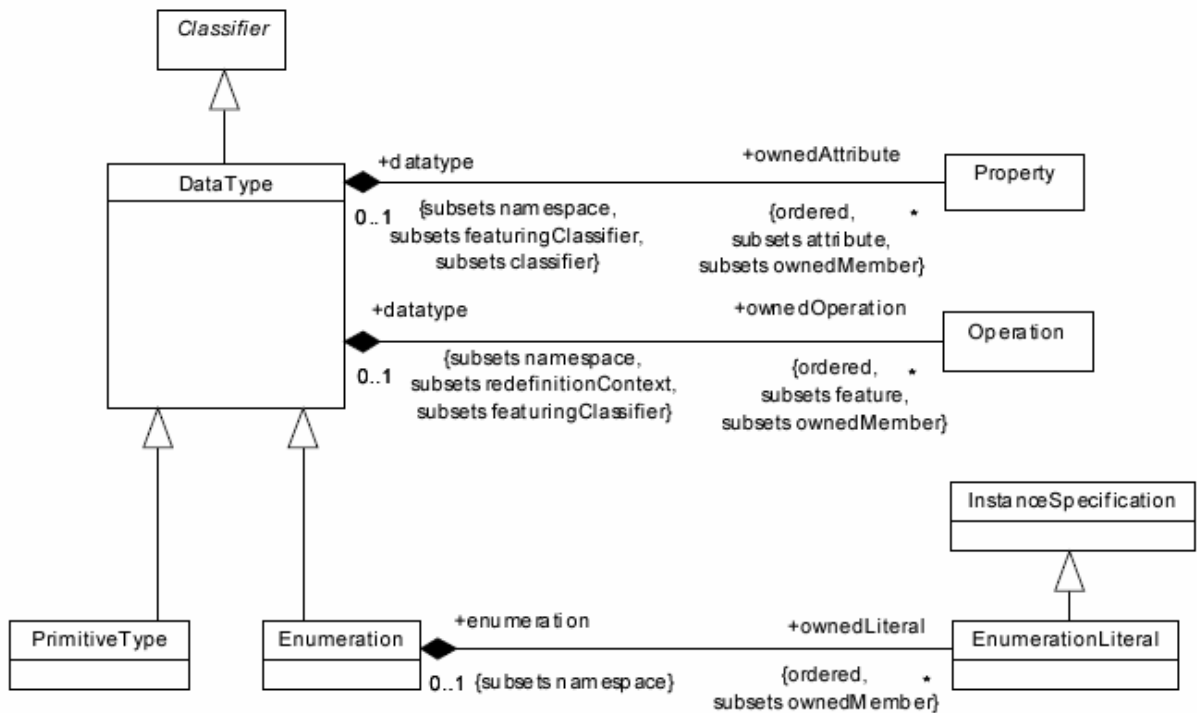


Figure 17 UML2 Data Type metaclasses

**3.4.2. Range (from Datatype::Datatypes)**

This class is not required; its features have been added to its subclasses. This is to allow a UML tool to work with standard types.

**3.4.3. BooleanType (from Datatype::Datatypes)**

BooleanType is represented by the <<booleanType>> stereotype which is a specialization of the <<primitiveType>> stereotype defined in Section 3.4.1. BooleanType is a concrete class and thus it is represented by a concrete stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 |                              | value       | Bool_Enum   | 1                   |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Name</b> | <b>Literals</b> |
|---|--------------------------------|-------------|-----------------|
| 1 |                                | Bool_Enum   | true, false     |

No semantic constraints are defined for this stereotype.

### 3.4.4. IntegerType (from Datatype::Datatypes)

IntegerType is represented by the <<integerType>> stereotype which is a specialization of the <<primitiveType>> stereotype defined in Section 3.4.1. IntegerType is a concrete class and thus it is represented by a concrete stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i> | <i>Type</i> | <i>Multiplicity</i> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 |                              | LOWER_LIMIT | Integer     | 1                   |
| 2 |                              | UPER_LIMIT  | Integer     | 1                   |

No semantic constraints are defined for this stereotype.

### 3.4.5. RealType (from Datatype::Datatypes)

RealType is represented by the <<realType>> stereotype which is a specialization of the <<primitiveType>> stereotypes defined in Section 3.4.1. RealType is a concrete class and thus it is represented by a concrete stereotype.

The attributes “encoding”, “allowNaN”, “LOWER-LIMIT” and “UPER-LIMIT” of RealType are represented by the corresponding properties added to the stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i> | <i>Type</i>   | <i>Multiplicity</i> |
|---|------------------------------|-------------|---------------|---------------------|
| 1 |                              | LOWER_LIMIT | Integer       | 1                   |
| 2 |                              | UPER_LIMIT  | Integer       | 1                   |
| 3 |                              | allowNaN    | Boolean       | 1                   |
| 4 |                              | Encoding    | Encoding_Enum | 1                   |

The following enumeration is required,

|   | <i>Enumeration Description</i> | <i>Type</i>   | <i>Literals</i> |
|---|--------------------------------|---------------|-----------------|
| 1 |                                | Encoding_Enum | Single, Double  |

### 3.4.6. OpaqueType (from Datatype::Datatypes)

OpaqueType is represented by the <<opaqueType>> stereotype which is a specialization of the <<primitiveType>> stereotype defined in Section 3.4.1. OpaqueType is a concrete class and thus it is represented by a concrete stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>  | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|--------------|-------------|---------------------|
| 1 | Type size In bits            | numberOfBits | Integer     | 1                   |

### 3.4.7. PrimitiveTypeWithSemantics (from DataType::Datatypes)

PrimitiveTypeWithSemantics is represented by the <<primitiveTypeWithSemantics>> stereotype which extends the DataType metaclass (from Kernel).

PrimitiveTypeWithSemantics is a concrete class and thus it is represented by a concrete stereotype.

Since this class links a primitive data type with its associated semantics, it must reference both.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>  | <b>Constrained feature</b> | <b>OCL Expression</b>            |
|---|--|----------------------------|----------------------------------|
| 1 | PrimitiveTypeWithSemantics string type attribute MUST reference an element stereotyped by <<primitiveType>> or its sub-stereotypes | not defined                | not defined                      |
| 2 | PrimitiveTypeWithSemantics MUST NOT own attributes   | DataType::ownedAttribute   | self.ownedAttribute ->size() = 0 |
| 3 | PrimitiveTypeWithSemantics MUST NOT own operations   | DataType::ownedOperation   | self.ownedOperation ->size() = 0 |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b>   | <b>Name</b>  | <b>Type</b>       | <b>Multiplicity</b> |
|---|--|--------------|-------------------|---------------------|
| 1 |  | invalidValue | ConstantPrimitive | 0..1                |
| 2 | type association. The type field should store the AUTOSAR shortname of the DataType.   | type         | String            | 1                   |
| 3 | CompuMethod association. Although we don't support this directly in the profile, we allow the fully qualified name to be entered for an already defined CompuMethod. | semantics    | String            | 1                   |

### 3.4.8. EnumerationType

EnumerationType is represented by the <<enumerationType>> stereotype which extends the Enumeration metaclass (from Kernel). EnumerationType is not available in the AUTOSAR metamodel. However, it can be used as a shortcut for PrimitiveTypeWithSemantics which represents an enumeration.

The physical values that are defined in the SW-COMPU-METHOD refer to the name of the owned EnumerationLiterals (from Kernel). The associated internal value is represented by a LiteralInteger. The EnumerationLiteral is stereotyped by the <<enumerationLiteral>> stereotype.

There are no semantic constraints defined.

### 3.5. Internal Behavior

These classes describe the interaction between an AtomicSoftwareComponent and the RTE.

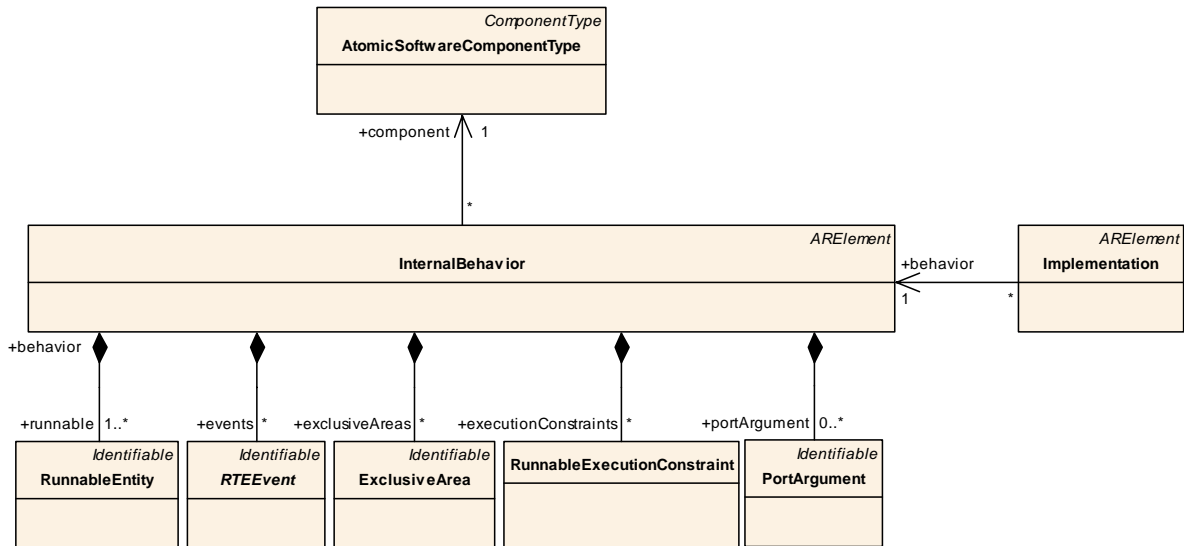


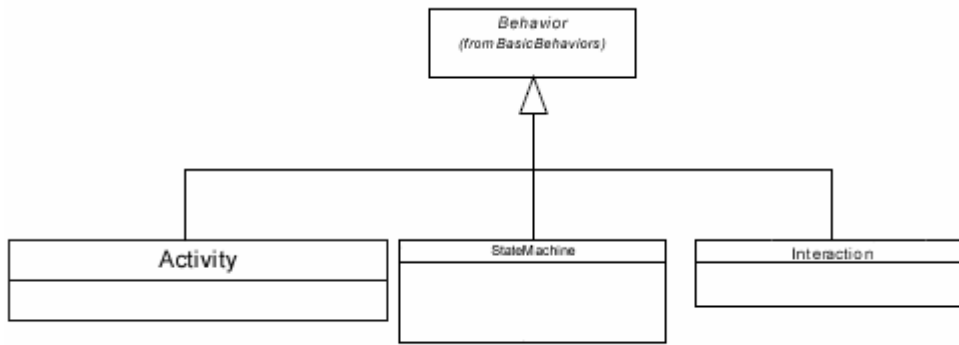
Figure 18 Internal Behavior

#### 3.5.1. InternalBehavior (from InternalBehavior)

InternalBehavior is represented by the <<internalBehavior>> stereotype which extends the Activity metaclass.

UML has many ways to describe behavior, all of which are specializations of the Behavior metaclass. Although this document will favor a description based on Activities (easiest mapping) it is also possible to use statemachines and interaction diagrams to essentially describe the same thing. It should be possible to transform between these different representations.

An InternalBehavior can only be associated with one AtomicSoftwareComponentType, to model this relationship the former will be an ownedElement of the latter.



**Figure 19 UML2 Behaviors**

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>                             | <b>Constrained feature</b> | <b>OCL Expression</b>                          |
|---|---|----------------------------|--|
| 1 | InternalBehaviour is owned by <<atomicSoftwareComponent>> | Kernel::owner              | self.owner.oclsTypeOf(AtomicSoftwareComponent) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>                   | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------------------------|-------------|---------------------|
| 1 |                              | supportsMultipleInstantiation | Boolean     | 1                   |

### 3.5.2. Implementation (from InternalBehavior:: Implementation)

Implementation is represented by the <<implementation>> stereotype which extends the Artifact metaclass.

This class is an umbrella for many low-level parameters of an implementation, which will be nested artifacts.

InternalBehavior, ProcessingUnit and Compiler are navigable associations from this class, these will be modeled using an Artifacts manifestation association. A Manifestation class will be implicitly created for these.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>      | <b>Constrained feature</b> | <b>OCL Expression</b>  |
|---|------------------------------------|----------------------------|--|
| 1 | No owned operations                | Artifact::ownedOperation   | self.ownedOperation->size() = 0  |
| 2 | No owned attributes                | Artifact::ownedAttribute   | self.ownedAttribute->size() = 0  |
| 3 | Must reference an InternalBehavior | Artifact::manifestation    | self.manifestation ->one( m   m.utilizedElement.oclsTypeOf(InternalBehavior) |
| 4 | Must reference a Compiler          | Artifact::manifestation    | self.manifestation ->one( m   m.utilizedElement.oclsTypeOf(Compiler)         |

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i>         | <i>Type</i>   | <i>Multiplicity</i> |
|---|------------------------------|---------------------|---------------|---------------------|
| 1 |                              | humanLanguage       | String        | 1                   |
| 2 |                              | programmingLanguage | ProgLang_Enum | 1                   |
| 3 |                              | codeGenerator       | String        | 1                   |
| 4 |                              | runtime             | String        | 1                   |
| 5 |                              | basicSW             | String        | 1                   |
| 6 |                              | requiredRTEVendor   | Boolean       | 1                   |

The following enumeration is required,

|   | <i>Enumeration Description</i> | <i>Type</i>   | <i>Literals</i> |
|---|--------------------------------|---------------|-----------------|
| 1 |                                | ProgLang_Enum | c, cpp, java    |

### 3.5.3. ResourceConsumption (from InternalBehavior::Implementation)

ResourceConsumption is represented by the <<resourceConsumption>> stereotype which extends the Artifact metaclass.

This class is an umbrella for many low-level parameters describing resource consumption; these will be nested artifacts.

The following semantic constraints are defined for this stereotype:

|   | <i>Constraint Description</i> | <i>Constrained feature</i> | <i>OCL Expression</i>                      |
|---|-------------------------------|----------------------------|--|
| 1 | Owned by an Implementation    | Kernel::owner              | self.owner.<br>oclIsTypeOf(Implementation) |
| 2 | No owned operations           | Artifact::ownedOperation   | self.ownedOperation->size() = 0            |
| 3 | No owned attributes           | Artifact::ownedAttribute   | self.ownedOperation->size() = 0            |

### 3.5.4. Code (from InternalBehavior::Implementation)

Code is represented by the <<code>> stereotype which extends the Artifact metaclass.

The following semantic constraints are defined for this stereotype:

|   | <i>Constraint Description</i>                        | <i>Constrained feature</i> | <i>OCL Expression</i>   |
|---|--|----------------------------|---|
| 1 | Owned by an Implementation                           | Kernel::owner              | self.owner.oclIsTypeOf(Implementation)  |
| 2 | No owned operations                                  | Artifact::ownedOperation   | self.ownedOperation->size() = 0   |
| 3 | Single attribute which must be of type CodeType_Enum | Artifact::ownedAttribute   | self.ownedOperation->size() = 1 and<br>self.ownedOperation->at[1]<br>.type.oclIsTypeOf(CodeType_Enum) |

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i> | <i>Type</i>   | <i>Multiplicity</i> |
|---|------------------------------|-------------|---------------|---------------------|
| 1 | Type of implementation       | type        | CodeType_Enum | 1                   |

The following enumeration is required,



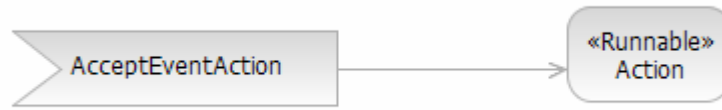


Figure 21 Activity diagram showing a model of runnable with its two actions and control flow

An AcceptEventAction can be triggered by multiple events and conceptually runs forever waiting for these.



Figure 22 UML2 AcceptEventAction metaclass

The following semantic constraints are defined for this stereotype (CallBehaviorAction)

|   | <b>Constraint Description</b>                               | <b>Constrained feature</b> | <b>OCL Expression</b>                                       |
|---|---|----------------------------|---|
| 1 | Must have an inbound control flow from an AcceptEventAction | ActivityNode::incoming     | self.incoming->one( ae   ae.outgoing.oclIsTypeOf(Runnable)) |

The following semantic constraints are defined for this stereotype (AcceptEventAction)

|   | <b>Constraint Description</b>                                | <b>Constrained feature</b> | <b>OCL Expression</b>                                       |
|---|--|----------------------------|---|
| 1 | Must have an inbound control flow from an CallBehaviorAction | ActivityNode::outgoing     | self.outgoing->one( ae   ae.incoming.oclIsTypeOf(Runnable)) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 |                              | symbol      | String      | 1                   |

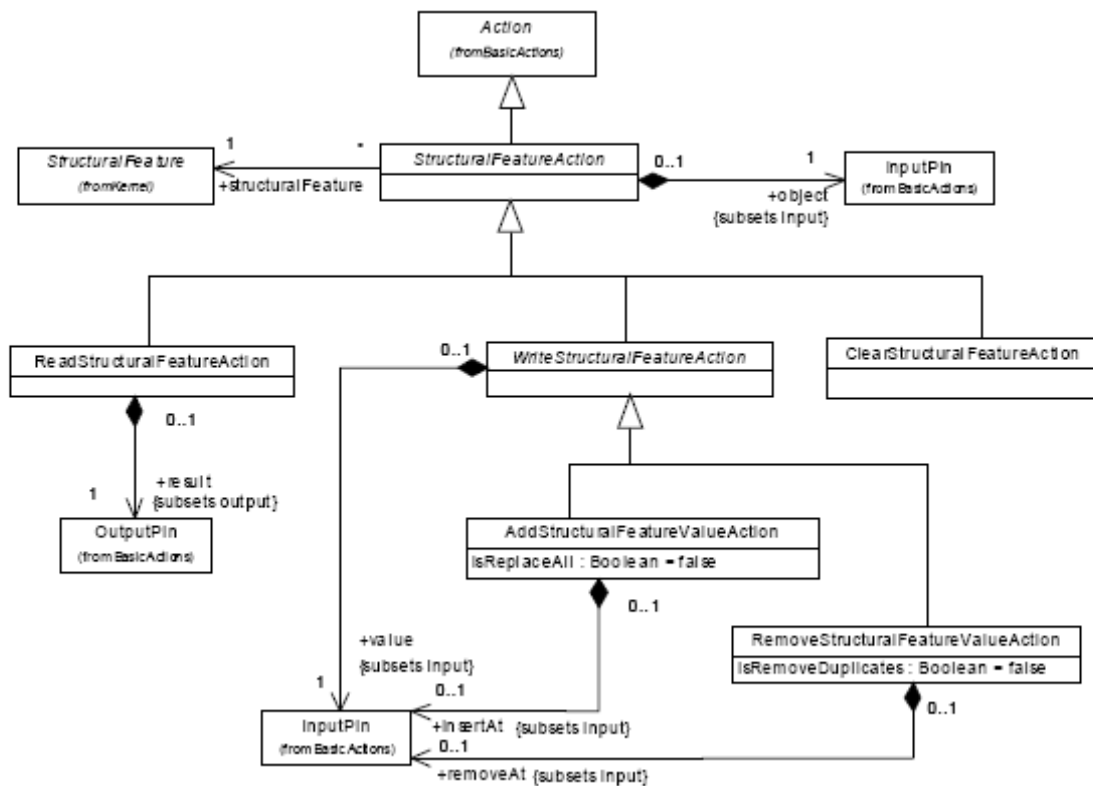
For Notation, see [10] 3.3.8.1.

**3.5.6. Data access**

Runnables are allowed to consume and produce data which may be associated with ports or shared attributes.

Data access for ports can either be implicit (for cat. I Runnable only) or can be explicitly initiated whilst running by an API call (cat. II).

For modeling the reading and writing of DataElementPrototypes the most appropriate UML actions are those derived from StructuralFeatureAction, see Figure 23 UML2 Structural Feature Actions. This allows an association to a port (the structural feature) giving context, the DataElement itself will be identified as a string attribute owned by the stereotype.



**Figure 23 UML2 Structural Feature Actions**

**3.5.6.1. DataReadAccess (from InternalBehavior:: DataElements)**

DataReadAccess is represented by the <<dataReadAccess>> stereotype which extends the ReadStructuralFeatureAction metaclass.

When a runnable is triggered it can specify that it requires read access to a data element on a port. To model this access a ReadStructuralFeatureAction will be used. The object InputPin is left unconnected, the result OutputPin should be connected to the Runnable and may be optionally typed with the PortInterface which owns the DataElement.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>  | <b>Constrained feature</b>                 | <b>OCL Expression</b>   |
|---|--|--|---|
| 1 | result pin should be connected using an ObjectFlow to a <<runnableEntity>>                 | ReadStructuralFeatureAction::result        | self.result.outgoing.target.owner.ocllsTypeOf(runnableEntity)     |
| 2 | structuralFeature should be associated with a Port owned by an <<atomicSoftwareComponent>> | StructuralFeatureAction::structuralFeature | self.structuralFeature.owner.ocllsTypeOf(atomicSoftwareComponent) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 | DataElement name             | name        | String      | 1                   |

For Notation, see [10] 3.3.10.1.

### 3.5.6.2. DataWriteAccess (from InternalBehavior:: DataElements)

DataWriteAccess is represented by the <<dataWriteAccess>> stereotype which extends the AddStructuralFeatureValueAction metaclass.

When a runnable (Cat I) finishes it can write its results to a data element. To model this access an AddStructuralFeatureValueAction will be used. The object InputPin is left unconnected, the value InputPin should be connected to the Runnable and may be optionally typed with the PortInterface which owns the DataElement.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>  | <b>Constrained feature</b>                 | <b>OCL Expression</b>   |
|---|--|--|---|
| 1 | value pin should be connected using an ObjectFlow to a <<runnableEntity>>                  | ReadStructuralFeatureAction::value         | self.value.incomming.source.owner.ocllsTypeOf(runnableEntity)     |
| 2 | structuralFeature should be associated with a Port owned by an <<atomicSoftwareComponent>> | StructuralFeatureAction::structuralFeature | self.structuralFeature.owner.ocllsTypeOf(atomicSoftwareComponent) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 | DataElement name             | name        | String      | 1                   |

For Notation, see [10] 3.3.10.1.

### 3.5.6.3. DataReceivePoint (from InternalBehavior:: DataElements)

DataReceivePoint is represented by the <<dataReceivePoint>> stereotype which extends the ReadStructuralFeatureAction metaclass.

If a Runnable wishes to read an attribute during execution it uses a DataReceivePoint which causes the RTE Generator to create an API call. The end result it similar to DataReadAccess it's just that the access mechanism is slightly different. To model this access a ReadStructuralFeatureAction will be used. The

object InputPin is left unconnected, the result OutputPin should be connected to the Runnable and may be optionally typed with the PortInterface which owns the DataElement.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>  | <b>Constrained feature</b>                 | <b>OCL Expression</b>   |
|---|--|--|---|
| 1 | result pin should be connected using an ObjectFlow to a <<runnableEntity>>                 | ReadStructuralFeatureAction::result        | self.result.outgoing.target.owner.oclIsTypeOf(runnableEntity)     |
| 2 | structuralFeature should be associated with a Port owned by an <<atomicSoftwareComponent>> | StructuralFeatureAction::structuralFeature | self.structuralFeature.owner.oclIsTypeOf(atomicSoftwareComponent) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 | DataElement name             | name        | String      | 1                   |

For Notation see [10] 3.3.10.1.

#### 3.5.6.4. DataSendPoint (from InternalBehavior:: DataElements)

DataSendPoint is represented by the <<dataSendPoint>> stereotype which extends the AddStructuralFeatureValueAction metaclass.

If a Runnable wishes to write to an attribute during execution it uses a DataSendPoint which causes the RTE Generator to create an API call. The end result is similar to DataWriteAccess it's just that the access mechanism is slightly different. To model this access an AddStructuralFeatureValueAction will be used. The object InputPin is left unconnected, the value InputPin should be connected to the Runnable and may be optionally typed with the PortInterface which owns the DataElement.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>  | <b>Constrained feature</b>                 | <b>OCL Expression</b>   |
|---|--|--|---|
| 1 | value pin should be connected using an ObjectFlow to a <<runnableEntity>>                  | ReadStructuralFeatureAction::value         | self.value.incoming.source.owner.oclIsTypeOf(runnableEntity)      |
| 2 | structuralFeature should be associated with a Port owned by an <<atomicSoftwareComponent>> | StructuralFeatureAction::structuralFeature | self.structuralFeature.owner.oclIsTypeOf(atomicSoftwareComponent) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 | DataElement name             | name        | String      | 1                   |

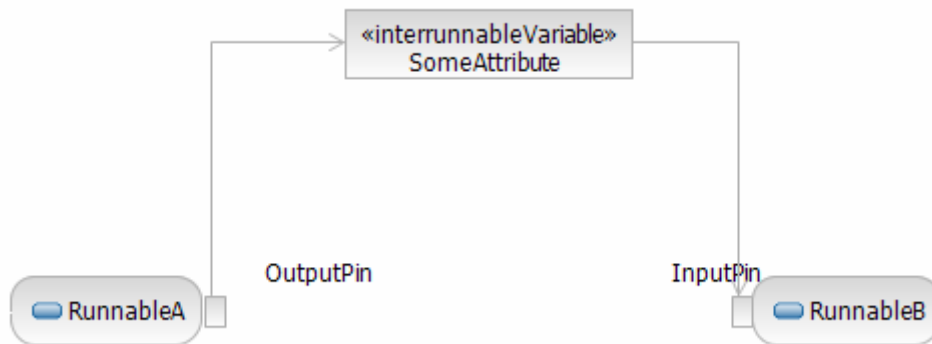
For Notation, see [10] 3.3.10.1.

**3.5.6.5. InterrunnableVariable**

InterrunnableVariable is represented by the <<interrunnableVariable>> stereotype which extends the ObjectNode metaclass.

An InterrunnableVariable can be read from/written to by RunnableEntities. To allow the former to be visible on an activity diagram it will be modeled as an ObjectNode, the type will be modeled as a string attribute.

To model the read and write access an ObjectFlow will be drawn either to or from pins on the RunnableEntity, see Figure 24 RunnableA has write access, RunnableB read access to an attribute.



**Figure 24 RunnableA has write access, RunnableB read access to an attribute.**

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>   | <b>Constrained feature</b> | <b>OCL Expression</b>                                  |
|---|---|----------------------------|--|
| 1 | Must have inbound ObjectFlow from a Pin owned by a <<runnableEntity>> | ActivityEdge::incoming     | self.incoming.source.owner.ocllsKindOf(runnableEntity) |
| 2 | Must have outbound ObjectFlow to a Pin owned by a <<runnableEntity>>  | ActivityEdge::outgoing     | self.outgoing.target.owner.ocllsKindOf(runnableEntity) |

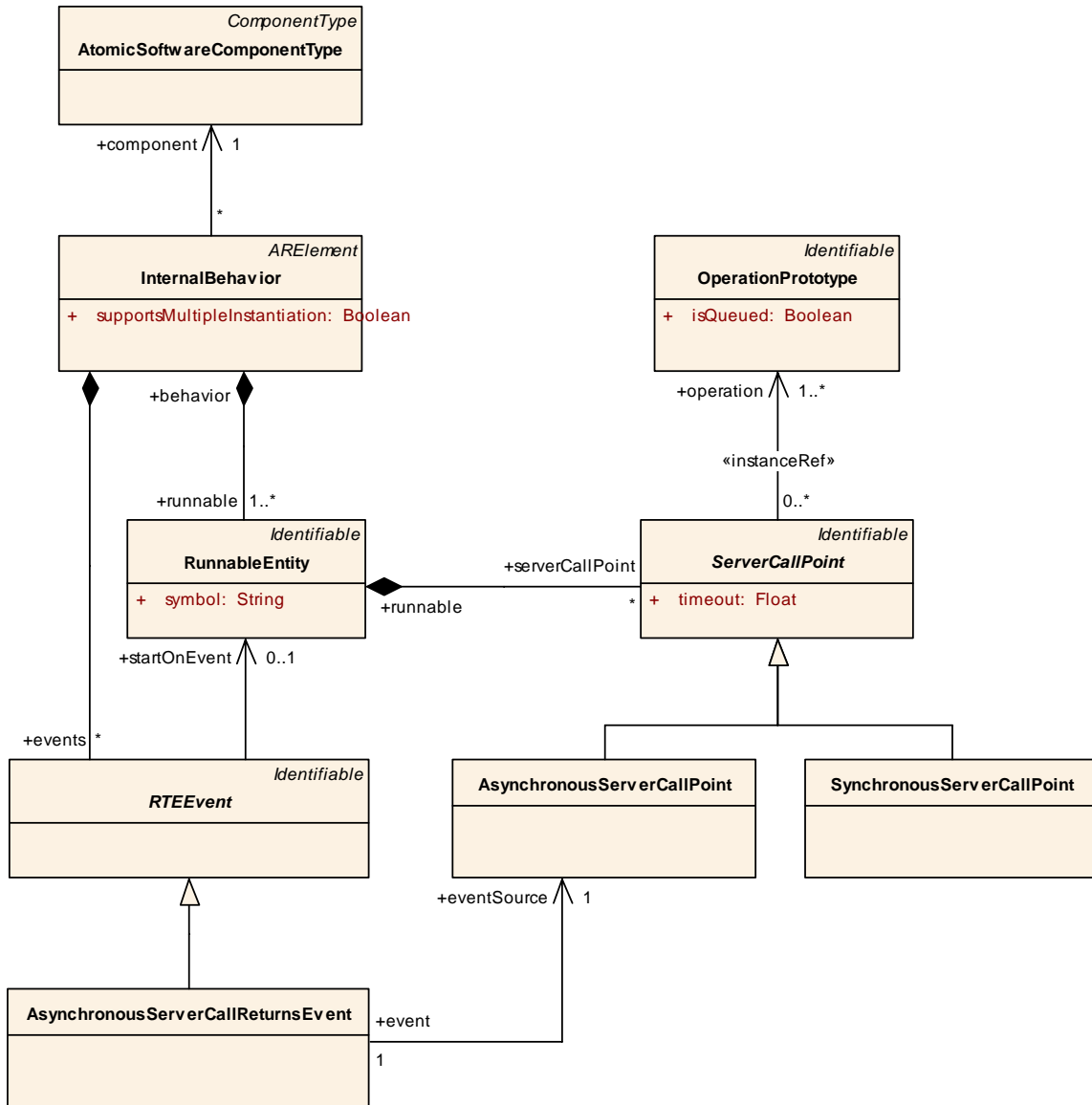
The following attributes are required for this stereotype

|   | <b>Attribute Description</b>  | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|---|-------------|-------------|---------------------|
| 1 | Type of the interrunnablevariable in the form (/model/package/./package/type) | Type        | String      | 1                   |
| 2 | Initial value. In the form of (/model/package/./package/ValueSpecification)   | initValue   | String      | 1                   |

For Notation, see [10] 3.3.9.1.

**3.5.6.6. ServerCallPoint (from InternalBehavior::ServerCall)**

ServerCallPoint is represented by the <<serverCallPoint>> stereotype which extends the CallOperationAction metaclass.



**Figure 25 Model of server call point**

A Runnable that wants to invoke an operation on a remote component must specify a ServerCallPoint which is associated to an OperationPrototype on a RequiredPort. For UML this can be represented as a CallOperationAction. A UML ControlFlow will be required from the calling <<runnableEntity>> to the ServerCallPoint.

The <<instanceRef>> between this event and the OperationPrototype will be modeled using a combination of a string attribute holding the port name, and CallOperationAction's operation and target associations. The former will reference

the operation and the latter the AtomicSoftwareComponent which owns the port. These will allow an unambiguous reference to be formed.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>  | <b>Constrained feature</b>                     | <b>OCL Expression</b>                            |
|---|--|--|--|
| 1 | ControlFlow which targets this class must source from a <<runnableEntity>> | ActivityEdge::incoming                         | self.incoming.source.ocllsKindOf(runnableEntity> |
| 2 | ControlFlow sourced from this class must target a <<runnableEntity>>       | ActivityEdge::outgoing                         | self.outgoing.target.ocllsKindOf(runnableEntity> |
| 3 | Incoming and Outgoing controlFlow must be from the same Element            | ActivityEdge::incoming, ActivityEdge::outgoing | self.outgoing.target = self.incoming.source      |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b>                   | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|--|-------------|-------------|---------------------|
| 1 | Name of the local port which has the operation | port        | String      | 1                   |
| 2 | Name of the operation                          | operation   | String      | 1                   |
| 3 |  | Timeout     | Float       | 1                   |

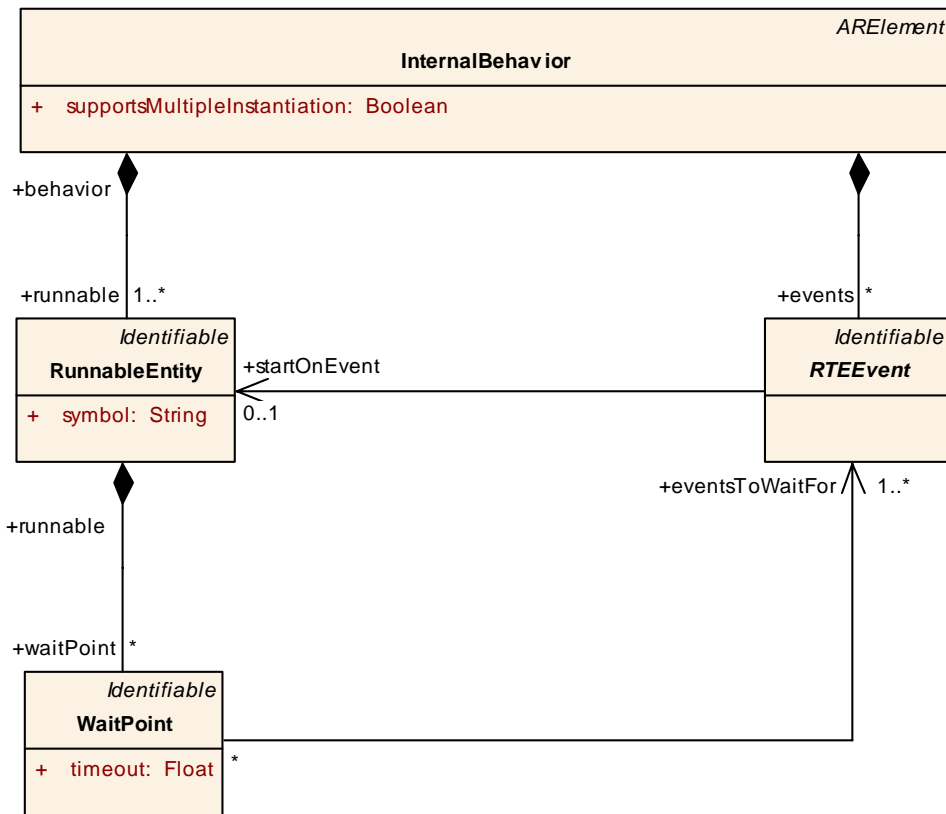
### 3.5.7. RTE Events

During execution, several run-time events will occur, such as the reception of a remote operation-invocation on a P-Port or a timeout on an R-Port that is not receiving the data-elements it expects. Describing an RTEEvent in the software-component template includes two aspects:

1. defining an event
2. defining how the RTE should deal with the event when it occurs

As described in the virtual functional bus specification, the implementation of a software-component can interact with the occurrence of such events in two ways:

- the RTE can be instructed to enable a specific runnable when the event occurs
- the RTE can provide "wait-points", that allow a runnable to block until an event in a set of events occurs



**Figure 26 Internal Behavior**

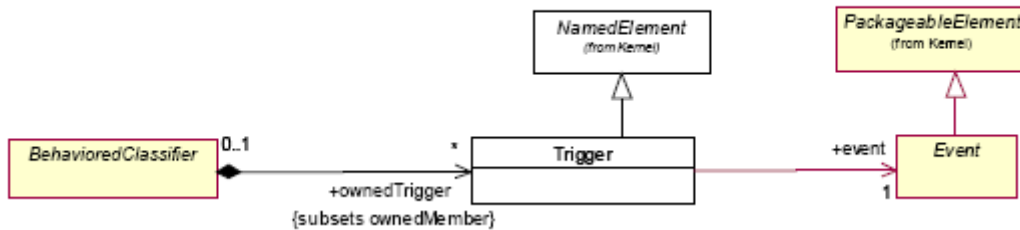
InternalBehavior is modeled as an ownedElement of an AtomicSoftwareComponentType, therefore RTEEvents are able to locally reference ports (to find DataElements and Operations) and not require a fully qualified name.

**3.5.7.1. RTEEvent (from InternalBehavior:: RTEEvents)**

RTEEvent is represented by the <<RTEEvent>> stereotype which extends the Event and Trigger metaclasses. For each RTEEvent a UML Event and Trigger are required. This class is abstract.

AUTOSAR events can enable runnables directly without the need for triggers. UML 2.0 relates an event to the invocation of behavior through a trigger hence a trigger will have to be implicitly created for each event.

For context, DataReceiveEvent and OperationInvokedEvent will have their Trigger associated with the port which owns the DataElement and OperationPrototype respectively; a stereotype attribute will then be required to resolve down to the individual element.



**Figure 27: UML 2.0 Event and Trigger relationship**

The following semantic constraints are defined for this stereotype (Trigger)

|   | <b>Constraint Description</b>   | <b>Constrained feature</b> | <b>OCL Expression</b>  |
|---|---|----------------------------|--|
| 1 | Trigger must reference an event stereotyped with <<RTEEvent>> or its descendants. | Trigger::event             | self.event.oclIsKindOf(rTEEvent)   |
| 2 | Trigger and Event owned by same element, the <<internalBehavior>>                 | Element::owner             | self.owner = self.event.owner and self.owner.oclIsTypeOf(internalBehavior) |

**3.5.7.2. DataReceivedEvent (from InternalBehavior:: RTEEvents)**

DataReceivedEvent is represented by the <<dataReceivedEvent>> stereotype which is a specialization of the <<RTEEvent>> stereotype.

The <<instanceRef>> between this event and the DataElementPrototype will be modeled using a combination of the Triggers port and a string attribute for the DataElement’s name.

The following semantic constraints are defined for this stereotype (Trigger)

|   | <b>Constraint Description</b>   | <b>Constrained feature</b> | <b>OCL Expression</b>              |
|---|---|----------------------------|------------------------------------|
| 1 | Trigger’s port associated with same owning AtomicSoftwareComponent as Trigger | Trigger::port              | self.port.owner = self.owner.owner |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b>                                     | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|--|-------------|-------------|---------------------|
| 1 | name of the data element corresponding to its NamedElement::name | data        | String      | 1                   |

### 3.5.7.3. OperationInvokedEvent (from InternalBehavior:: RTEEvents)

OperationInvokedEvent is represented by the <<operationInvokedEvent>> stereotype which is a specialization of the <<RTEEvent>> stereotype.

The <<instanceRef>> between this event and the OperationPrototype will be modeled using a combination of the triggers port association and a string attribute for the Operation's name.

The following semantic constraints are defined for this stereotype (Trigger)

|   | <b>Constraint Description</b>   | <b>Constrained feature</b> | <b>OCL Expression</b>              |
|---|---|----------------------------|------------------------------------|
| 1 | Trigger's port associated with same owning AtomicSoftwareComponent as Trigger | Trigger::port              | self.port.owner = self.owner.owner |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b>                                  | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|---|-------------|-------------|---------------------|
| 1 | name of the operation corresponding to its NamedElement::name | data        | String      | 1                   |

### 3.5.7.4. DataSendCompletedEvent (from InternalBehavior:: RTEEvents)

DataSendCompletedEvent is represented by the <<dataSendCompletedEvent>> stereotype which is a specialization of the <<RTEEvent>> stereotype.

The reference to the DataSendPoint will be modeled as a string attribute which will hold its name.

No additional constraints are defined for this stereotype.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b>                                  | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|---|-------------|-------------|---------------------|
| 1 | Data send point name; corresponding to its NamedElement::name | eventSource | String      | 1                   |

### 3.5.7.5. TimingEvent (from InternalBehavior:: RTEEvents)

TimingEvent is represented by the <<timingEvent>> stereotype which extends the TimeEvent metaclass.

TimingEvent allows a runnable to be started periodically. The UML2 TimeEvent allows for behavior to be executed at some point in time (absolute or relative) so the timingEvent stereotype will be a self perpetuating event.

No additional constraints are defined for this stereotype.

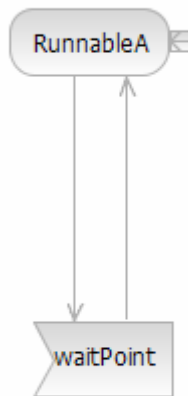
The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>  | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|--------------|-------------|---------------------|
| 1 | Period of timer              | timingPeriod | Float       | 1                   |

**3.5.7.6. WaitPoint (from InternalBehavior:: RTEEvents)**

WaitPoint is represented by the <<waitPoint>> stereotype which extends the AcceptEventAction metaclass.

A WaitPoint allows a runnable to block waiting for an event. To model this in UML an AcceptEventAction will be used with two control flows to and from the RunnableEntity.



**Figure 28 UML Activity representation of WaitPoint**

An AcceptEventAction can have one or more triggers, so only one WaitPoint is required per-Runnable.

The following semantic constraints are defined for this stereotype:

|   | <b>Constraint Description</b>   | <b>Constrained feature</b> | <b>OCL Expression</b>                           |
|---|---|----------------------------|---|
| 1 | ControlFlow outgoing from this Activity must terminate at a CallBehaviorAction with the <<runnableEntity>> stereotype | ActivityNode::outgoing     | self.outgoing.target.oclsKindOf(runnableEntity) |
| 2 | ControlFlow incoming to this Activity must originate from a CallBehaviorAction with the <<runnableEntity>> stereotype | ActivityNode::incoming     | self.incoming.source.oclsKindOf(runnableEntity) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 |                              | timeout     | Float       | 0..1                |

## 4. ECU Model

This part of the AUTOSAR metamodel is intended for modeling ECU hardware instances. It allows UML types to represent ECU hardware instances and support AUTOSAR ECU modeling.

AUTOSAR ECU instances are different to UML instances because they allow structural changes (ports) to be added, thus UML instances and diagrams are not sufficient. It has therefore been decided to model ECU HWElements as stereotyped classes enforcing instance like semantics using constraints.

### 4.1. ECUResourceTemplate – General Elements

In the ECU Resource Template the General HW Element serves as an abstract base class to describe properties common for all specific HW Elements.

#### 4.1.1. HWElement (from ECUResourceTemplate)

A HWElement is used to describe common properties for all hardware elements.

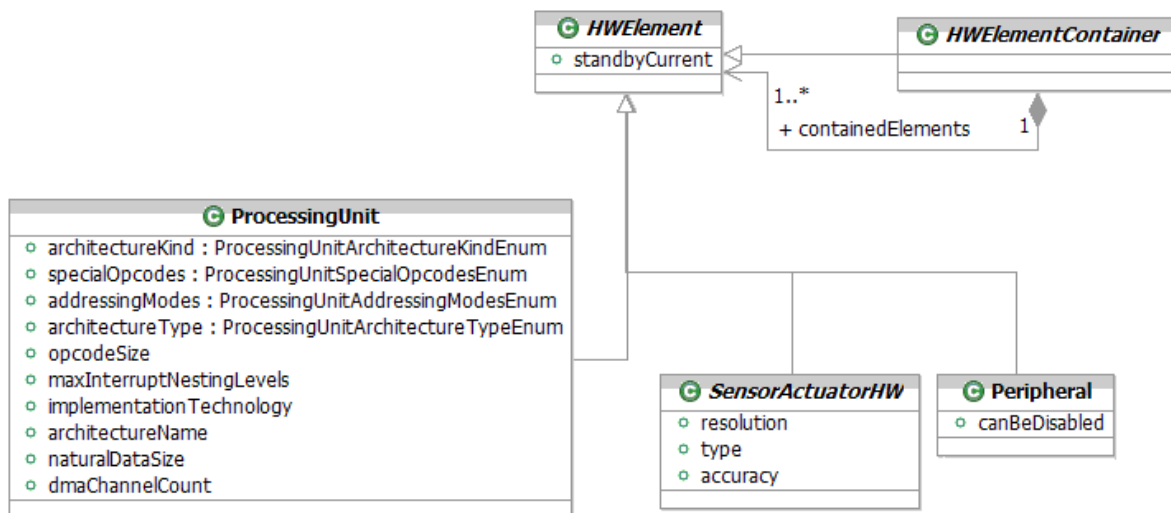


Figure 29 HWElements

A HWElement is represented by the abstract <<HWElement>> stereotype which extends the Property and Class metaclasses (from Kernel).

HWElement represent instances of electronic components which have ports. A HWElementContainer (HWElement subclass) may also own other HWElements. To model this in UML the container will be a structured class, the contained electronic components will be parts. UML Properties cannot own ports; therefore for each part a type will be required to own the ports, hence HWElement extends both

Property and Class. If a HWElement doesn't require ports a corresponding type is not required.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>   | <b>Constrained feature</b>        | <b>OCL Expression</b>   |
|---|---|-----------------------------------|---|
| 1 | Property with <<HWElement>> must have type either set to Class with <<HWElement>> or left empty | TypedElement::type                | self.type implies self.type.ocllsTypeOf(HWElement)                        |
| 2 | Owned ports must be stereotyped with <<HWPport>> or decedents.                                  | EncapsulatedClassifier::ownedPort | Self.ownedPort->forAll( p   p.type.ocllsTypeOf(HWPport))                  |
| 3 | Class with <<HWElement>> stereotype should be owned either by <<ECU>> or <<HWElementContainer>> | Element::owner                    | self.owner.ocllsTypeOf(ECU) or self.owner.ocllsTypeOf(HWElementContainer) |

#### 4.1.2. HWElementContainer (from ECUResourceTemplate)

This class is not required. It is abstract and provides containment support for HWElements – this is something the HWElement stereotype implicitly supports through its extension of UML2 Class.

#### 4.1.3. ECU (from ECUResourceTemplate)

The ECU (Electronic Control Unit) is represented by the <<ECU>> stereotype which is a specialization of the <<HWElement>> stereotype.

This class inherits the UML2 structural features from its parent (HWElement) which provides support for structural features (parts & ports).

4.1.4. HWPort (from ECUResourceTemplate)

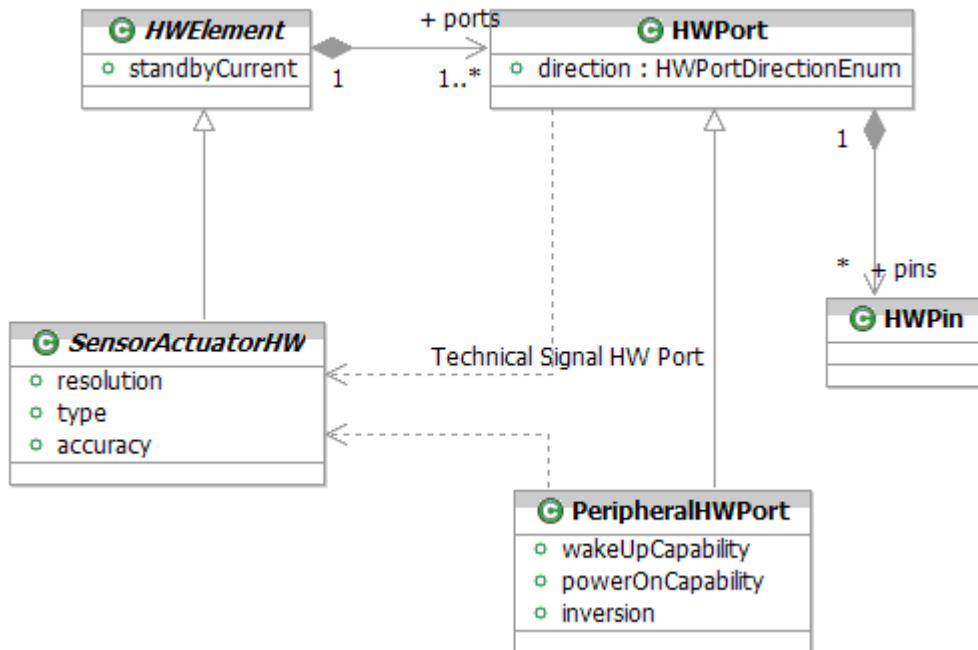


Figure 30 HWPorts

HWPort is represented by the <<HWPort>> stereotype which extends the Port metaclass.

This will require tagged values to represent the attribute direction and HWPin.

HWPort is a property of an ECU and can be connected to other ports. This fits well with the Port metaclass.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b> | <b>OCL Expression</b>             |
|---|-------------------------------|----------------------------|-----------------------------------|
| 1 | Owned by <<HWElement>>        | Element::owner             | self.owner.oclIsKindOf(HWElement) |

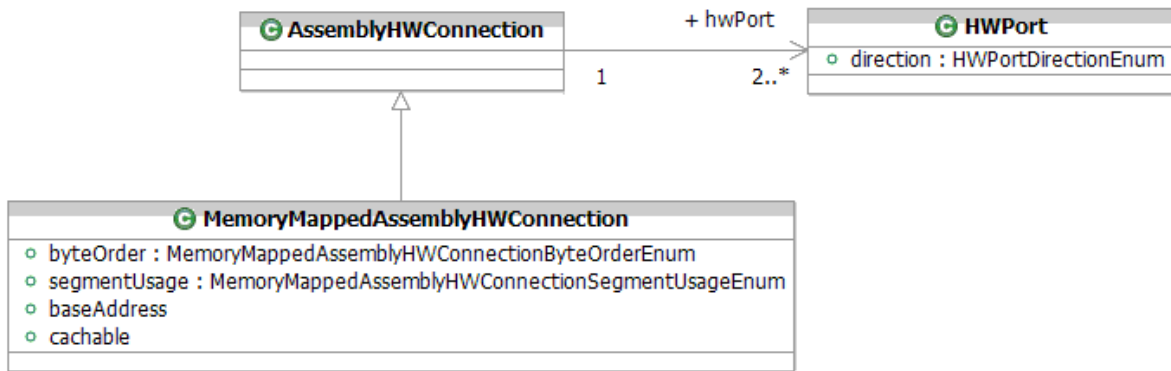
The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b>     | <b>Multiplicity</b> |
|---|------------------------------|-------------|-----------------|---------------------|
| 1 | Direction of comms           | direction   | HW_PortDir_Enum | 1                   |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Type</b>     | <b>Literals</b> |
|---|--------------------------------|-----------------|-----------------|
| 1 |                                | HW_PortDir_Enum | in, out, inout  |

**4.1.5. AssemblyHWConnection (from ECUResourceTemplate)**



**Figure 31 HWConnectors**

AssemblyHWConnection is represented by the <<assemblyHWConnection>> stereotype which extends the Connector metaclass.

AssemblyHWConnection is used to connect hardware ports together, in a similar way to the UML Connector which connects Ports.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>     | <b>Constrained feature</b> | <b>OCL Expression</b>                            |
|---|-----------------------------------|----------------------------|--|
| 1 | Connector ends must be <<HWPort>> | ConnectorEnd::role         | self.end->forAll( e   e.role.oclsTypeOf(HWPort)) |

## 4.2. Sensor Actuator

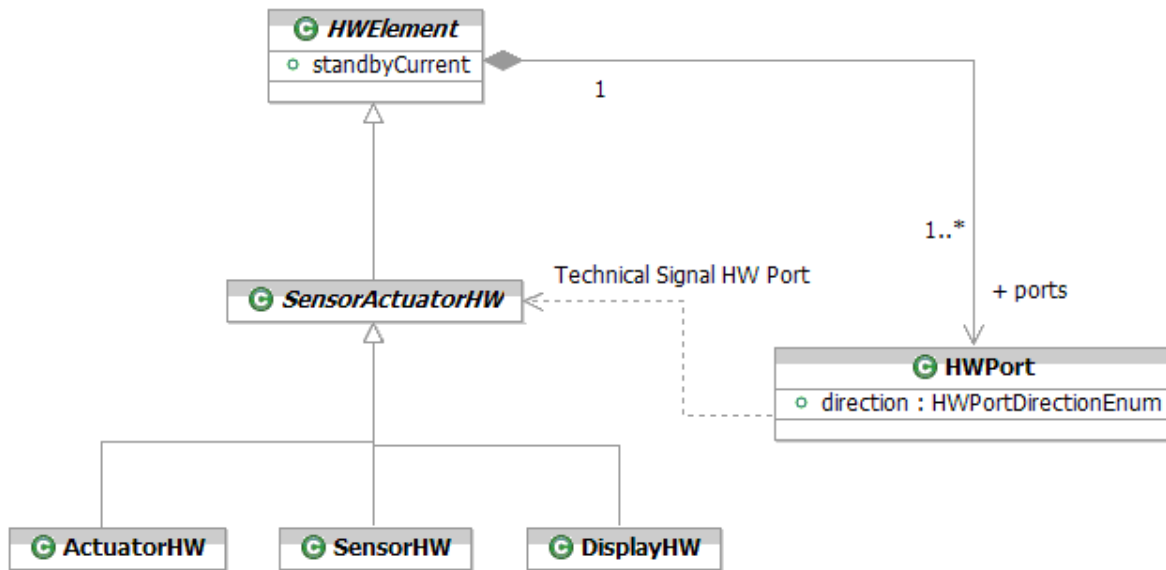


Figure 32 SensorActuators

### 4.2.1. SensorActuatorHW (from ECUResourceTemplate::Sensor Actuator)

The SensorActuatorHW is represented by the abstract <<sensorActuatorHW>> stereotype which is a specialization of the <<HWElement>> stereotype.

There are no semantic constraints defined.

### 4.2.2. ActuatorHW (from ECUResourceTemplate::Sensor Actuator)

The ActuatorHW is represented by the <<actuatorHW>> stereotype which is a specialization of the <<sensorActuatorHW>> stereotype.

There are no semantic constraints defined.

### 4.2.3. SensorHW (from ECUResourceTemplate::Sensor Actuator)

The SensorHW is represented by the <<sensorHW>> stereotype which is a specialization of the <<sensorActuatorHW>> stereotype.

There are no semantic constraints defined.

### 4.2.4. DisplayHW (from ECUResourceTemplate::Sensor Actuator)

The DisplayHW is represented by the <<displayHW>> stereotype which is a specialization of the <<sensorActuatorHW>> stereotype.

There are no semantic constraints defined.

## 4.3. Memory

### 4.3.1. MemoryMappedHWPort (from ECUResourceTemplate::Memory)

MemoryMappedHWPort is represented by the <<memoryMappedHWPort>> stereotype which is a specialization of the <<HWPort>> stereotype.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>           | <b>Constrained feature</b> | <b>OCL Expression</b>            |
|---|---|----------------------------|----------------------------------|
| 1 | Must be owned by a HWElement or subtype | Element::owner             | self.owner.oclsTypeOf(HWElement) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>        | <b>Type</b>   | <b>Multiplicity</b> |
|---|------------------------------|--------------------|---------------|---------------------|
| 1 | data width                   | dataInterfaceWidth | Integer       | 1                   |
| 2 | Port type                    | hwportType         | portType_Enum | 1                   |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Type</b>   | <b>Literals</b>          |
|---|--------------------------------|---------------|--------------------------|
| 1 |                                | portType_Enum | unspecified,data,control |

### 4.3.2. MemoryMappedAssemblyHWConnection (from ECUResourceTemplate::Memory)

MemoryMappedAssemblyHWConnection is represented by the <<memoryMappedAssemblyHWConnection>> stereotype which is a specialization of the <<assemblyHWConnection>> stereotype.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>                 | <b>Constrained feature</b> | <b>OCL Expression</b>  |
|---|---|----------------------------|--|
| 1 | Connector ends must be <<memoryMappedHWPort>> | ConnectorEnd::role         | self.end->forAll( e   e.role.oclsTypeOf(memoryMappedHWPort)) |

### 4.3.3. ProvidedMemorySegment (from ECUResourceTemplate::Memory)

The ProvidedMemorySegment is represented by the <<providedMemorySegment>> stereotype which is a specialization of the <<HWElement>> stereotype.

There are no semantic constraints defined.

#### 4.3.4. ProvidedNVMemorySegment (from ECUResourceTemplate::Memory)

The ProvidedNVMemorySegment is represented by the <<providedNVMemorySegment>> stereotype which is a specialization of the <<providedMemorySegment>> stereotype.

There are no semantic constraints defined.

### 4.4. Basic Elements

#### 4.4.1. ErrorDetectionCorrection (from ECUResourceTemplate::Basic Elements)

ErrorDetectionCorrection is represented by the <<errorDetectionCorrection>> stereotype which extends the Class metaclass.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>                    | <b>Constrained feature</b> | <b>OCL Expression</b>                            |
|---|--|----------------------------|--|
| 1 | Owned by a <<providedMemorySegment>> or subclass | Element::owner             | self.owner.oclsTypeOf(provide<br>dMemorySegment) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>   | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|---------------|-------------|---------------------|
| 1 |                              | extraBits     | Integer     | 1                   |
| 2 |                              | detectedBits  | Integer     | 1                   |
| 3 |                              | connectedBits | Integer     | 1                   |

### 4.5. ECU Electronics

#### 4.5.1. ECUElectronics (from ECUResourceTemplate::ECU Electronics)

The ECUElectronics is represented by the abstract <<ECUElectronics>> stereotype which is a specialization of the <<HWElement>> stereotype.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>                              | <b>Constrained feature</b>     | <b>OCL Expression</b>   |
|---|--|--------------------------------|---|
| 1 | Must be contained within <<ECU>> or <<HWElementContainer>> | Element::owner                 | self.owner.oclsTypeOf(ECU)<br>OR<br>self.owner.oclsTypeOf(HWEle<br>mentContainer) |
| 2 | Cannot contain other <<HWElement>>                         | StructuredClassifier::par<br>t | not self.part->exists(p  <br>p.type.oclsTypeOf(HWElemen<br>t))                    |

#### 4.5.2. Clock (from ECUResourceTemplate::ECU Electronics)

Clock is represented by the <<clock>> stereotype which is a specialization of the <<ECUElectronics>> stereotype.

No semantic constraints are defined.

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i>      | <i>Type</i>    | <i>Multiplicity</i> |
|---|------------------------------|------------------|----------------|---------------------|
| 1 |                              | type             | ClockType_Enum | 1                   |
| 2 |                              | adjustable       | Boolean        | 1                   |
| 3 |                              | clockStartupTime | Float          | 1                   |
| 4 |                              | clockTolerance   | Float          | 1                   |
| 5 |                              | clockJitter      | Float          | 1                   |

The following enumeration is required,

|   | <i>Enumeration Description</i> | <i>Type</i>    | <i>Literals</i> |
|---|--------------------------------|----------------|-----------------|
| 1 |                                | ClockType_Enum | PLL, divider    |

## 4.6. Peripherals

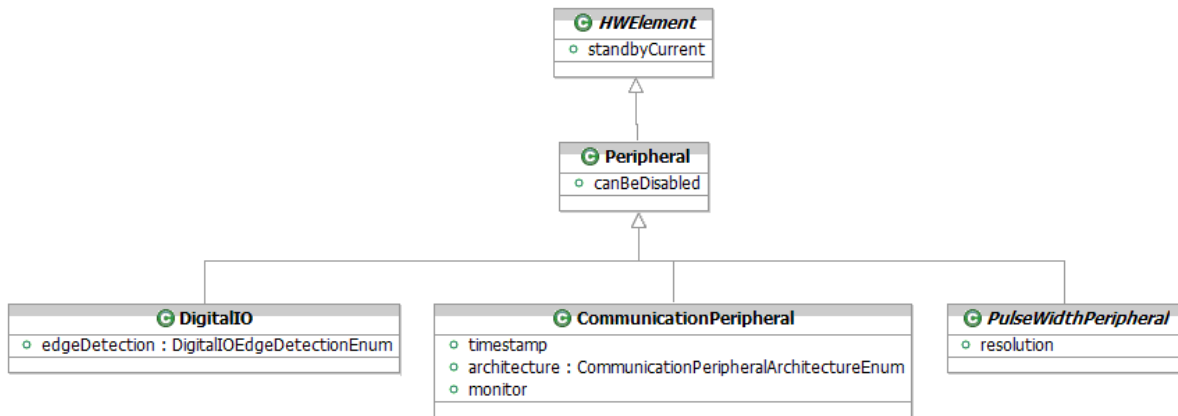


Figure 33 Peripherals

### 4.6.1. Peripheral (from ECUResourceTemplate::Peripherals)

The Peripheral is represented by the abstract <<peripheral>> stereotype which is a specialization of the <<HWElement>> stereotype.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>                              | <b>Constrained feature</b> | <b>OCL Expression</b>   |
|---|--|----------------------------|---|
| 1 | Must be contained within <<ECU>> or <<HWElementContainer>> | Element::owner             | self.owner.oclIsTypeOf(ECU) or self.owner.oclIsTypeOf(HWElementContainer) |
| 2 | Cannot contain other <<HWElement>>                         | StructuredClassifier::part | not self.part->exists(p   p.type.oclIsTypeOf(HWElement))                  |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>   | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|---------------|-------------|---------------------|
| 1 |                              | canBeDisabled | Boolean     | 1                   |

### 4.6.2. AnalogueIO (from ECUResourceTemplate::Peripherals)

AnalogueIO is represented by the abstract <<analogueIO>> stereotype which is a specialization of the <<peripheral>> stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i>    | <i>Type</i> | <i>Multiplicity</i> |
|---|------------------------------|----------------|-------------|---------------------|
| 1 |                              | resolution     | Integer     | 1                   |
| 2 |                              | accuracy       | Float       | 1                   |
| 3 |                              | mode           | Mode_Enum   | 1                   |
| 4 |                              | width          | Integer     | 1                   |
| 5 |                              | conversionTime | Float       | 0..1                |
| 6 |                              | multiplexedN   | Integer     | 0..1                |
| 7 |                              | multiplexedM   | Integer     | 0..1                |

The following enumeration is required,

|   | <i>Enumeration Description</i> | <i>Type</i> | <i>Literals</i>                |
|---|--------------------------------|-------------|--------------------------------|
| 1 |                                | Mode_Enum   | continuous, single, sequential |

#### 4.6.3. ADC (from ECUResourceTemplate::Peripherals)

ADC is represented by the <<ADC>> stereotype which is a specialization of the <<analogueIO>> stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i>     | <i>Type</i>   | <i>Multiplicity</i> |
|---|------------------------------|-----------------|---------------|---------------------|
| 1 |                              | behaviorAtLimit | Behavior_Enum | 1                   |
|   |                              |                 |               |                     |

The following enumeration is required,

|   | <i>Enumeration Description</i> | <i>Type</i>   | <i>Literals</i>                     |
|---|--------------------------------|---------------|-------------------------------------|
| 1 |                                | Behavior_Enum | clipping, default, undefined, other |

#### 4.6.4. DAC (from ECUResourceTemplate::Peripherals)

DAC is represented by the <<DAC>> stereotype which is a specialization of the <<analogueIO>> stereotype.

There are no semantic constraints defined.

#### 4.6.5. DigitalIO (from ECUResourceTemplate::Peripherals)

DigitalIO is represented by the <<digitalIO>> stereotype which is a specialization of the <<peripheral>> stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>   | <b>Type</b>      | <b>Multiplicity</b> |
|---|------------------------------|---------------|------------------|---------------------|
| 1 |                              | edgeDetection | digitalEdge_Enum | 1                   |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Type</b>      | <b>Literals</b>                |
|---|--------------------------------|------------------|--------------------------------|
| 1 |                                | digitalEdge_Enum | rising, falling, risingFalling |

#### 4.6.6. CommunicationPeripheral (from ECUResourceTemplate::Peripherals)

CommunicationPeripheral is represented by the <<communicationPeripheral>> stereotype which is a specialization of the <<peripheral>> stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>  | <b>Type</b>       | <b>Multiplicity</b> |
|---|------------------------------|--------------|-------------------|---------------------|
| 1 |                              | Architecture | Architecture_Enum | 1                   |
| 2 |                              | Monitor      | Boolean           | 1                   |
| 3 |                              | Timestamp    | Boolean           | 1                   |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Type</b>       | <b>Literals</b>            |
|---|--------------------------------|-------------------|----------------------------|
| 1 |                                | Architecture_Enum | master, slave, multiMaster |

#### 4.6.7. CommunicationProtocol (from ECUResourceTemplate::Peripherals)

CommunicationProtocol is represented by the <<communicationProtocol>> stereotype which is a specialization of the <<peripheral>> stereotype.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>          | <b>Constrained feature</b> | <b>OCL Expression</b>                           |
|---|--|----------------------------|---|
| 1 | Owned by a <<communicationPeripheral>> | Element::owner             | self.owner.ocllsTypeOf(communicationPeripheral) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 |                              | version     | String      | 0..1                |

#### 4.6.8. CommunicationFilter (from ECUResourceTemplate::Peripherals)

CommunicationFilter is represented by the <<communicationFilter>> stereotype which extends the Property metaclass.

This is a property of a Communication Peripheral.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>          | <b>Constrained feature</b> | <b>OCL Expression</b>                           |
|---|--|----------------------------|---|
| 1 | Owned by a <<communicationPeripheral>> | Kernel::ownedAttribute     | Self.class.isOclTypeOf(communicationPeripheral) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 |                              | filterWidth | Integer     | 1                   |
| 2 |                              | filterCount | Integer     | 1                   |
| 3 |                              | wildcards   | Boolean     | 1                   |

#### 4.6.9. PulseWidthPeripheral (from ECUResourceTemplate::Peripherals)

PulseWidthPeripheral is represented by the abstract <<pulseWidthPeripheral>> stereotype which is a specialization of the <<peripheral>> stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 |                              | Resolution  | Integer     | 1                   |

#### 4.6.10. PWD (from ECUResourceTemplate::Peripherals)

PWD is represented by the <<PWD>> stereotype which is a specialization of the <<pulseWidthPeripheral>> stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>   | <b>Type</b>   | <b>Multiplicity</b> |
|---|------------------------------|---------------|---------------|---------------------|
| 1 |                              | phaseDetector | Boolean       | 1                   |
| 2 |                              | Mode          | PWD_Mode_Enum | 1                   |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Type</b>   | <b>Literals</b>                |
|---|--------------------------------|---------------|--------------------------------|
| 1 |                                | PWD_Mode_Enum | twoPhaseInput, threePhaseInput |

#### 4.6.11. PWM (from ECUResourceTemplate::Peripherals)

PWM is represented by the <<PWM>> stereotype which is a specialization of the <<pulseWidthPeripheral>> stereotype.

There are no semantic constraints defined.

#### 4.6.12. Buffer (from ECUResourceTemplate::Peripherals)

Buffer is represented by the <<buffer>> stereotype which extends the Property metaclass.

This is a property of a PeripheralHWPport.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>    | <b>Constrained feature</b> | <b>OCL Expression</b>                    |
|---|----------------------------------|----------------------------|--|
| 1 | Owned by a <<peripheralHWPport>> | Element::owner             | self.owner.oclsTypeOf(peripheralHWPport) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 |                              | Number      | Boolean     | 1                   |
| 2 |                              | Type        | Buffer_Enum | 1                   |
| 3 |                              | Size        | Integer     | 1                   |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Type</b> | <b>Literals</b>      |
|---|--------------------------------|-------------|----------------------|
| 1 |                                | Buffer_Enum | Input, output, inout |

#### 4.6.13. PeripheralHWPport (from ECUResourceTemplate::Peripherals)

PeripheralHWPport is represented by the <<peripheralHWPport>> stereotype which is a specialization of the <<HWPport>> stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>       | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------------|-------------|---------------------|
| 1 |                              | Inversion         | Boolean     | 0..1                |
| 2 |                              | wakeUpCapability  | Boolean     | 0..1                |
| 3 |                              | powerOnCapability | Boolean     | 0..1                |

### 4.7. ProcessingUnit

#### 4.7.1. ProcessingUnit (from ECUResourceTemplate::Processing Unit)

The ProcessingUnit is represented by the abstract <<processingUnit>> stereotype which is a specialization of the <<HWElement>> stereotype.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>                              | <b>Constrained feature</b> | <b>OCL Expression</b>   |
|---|--|----------------------------|---|
| 1 | Must be contained within <<ECU>> or <<HWElementContainer>> | Element::owner             | self.owner.ocllsTypeOf(ECU) or self.owner.ocllsTypeOf(HWElementContainer) |
| 2 | Cannot contain other <<HWElement>>                         | StructuredClassifier::part | not self.part->exists(p   p.type.ocllsTypeOf(HWElement))                  |

The following attributes are required for this stereotype

|    | <b>Attribute Description</b> | <b>Name</b>               | <b>Type</b>           | <b>Multiplicity</b> |
|----|------------------------------|---------------------------|-----------------------|---------------------|
| 1  |                              | architectureKind          | ArchitectureKind_Enum | 0..1                |
| 2  |                              | architectureName          | String                | 0..1                |
| 3  |                              | architectureType          | ArchitectureType_Enum | 0..1                |
| 4  |                              | implementationTechnology  | String                | 0..1                |
| 5  |                              | naturalDataSize           | Integer               | 1                   |
| 6  |                              | addressingModes           | A_Mode_Enum           | 0..*                |
| 7  |                              | opcodeSize                | String                | 0..1                |
| 8  |                              | specialOpCodes            | OpCodes_Enum          | 0..*                |
| 9  |                              | dmaChannelCount           | Integer               | 0..1                |
| 10 |                              | maxInterruptNestingLevels | Integer               | 0..1                |

The following enumerations are required,

|   | <b>Enumeration Description</b> | <b>Type</b>           | <b>Literals</b>                       |
|---|--------------------------------|-----------------------|---------------------------------------|
| 1 |                                | ArchitectureKind_Enum | HARVARD, vNEUMAN                      |
| 2 |                                | ArchitectureType_Enum | RISC, CISC, SIMD, MIMD                |
| 3 |                                | A_Mode_Enum           | direct, indirect, indexed             |
| 4 |                                | OpCodes_Enum          | bitManipulation, MUL, DIV, MAC, Float |

## 5. System Modeling

This section describes the system model which defines the system topology, communication frame structure and the mapping of atomic software components to ECUs.

### 5.1. Top Level

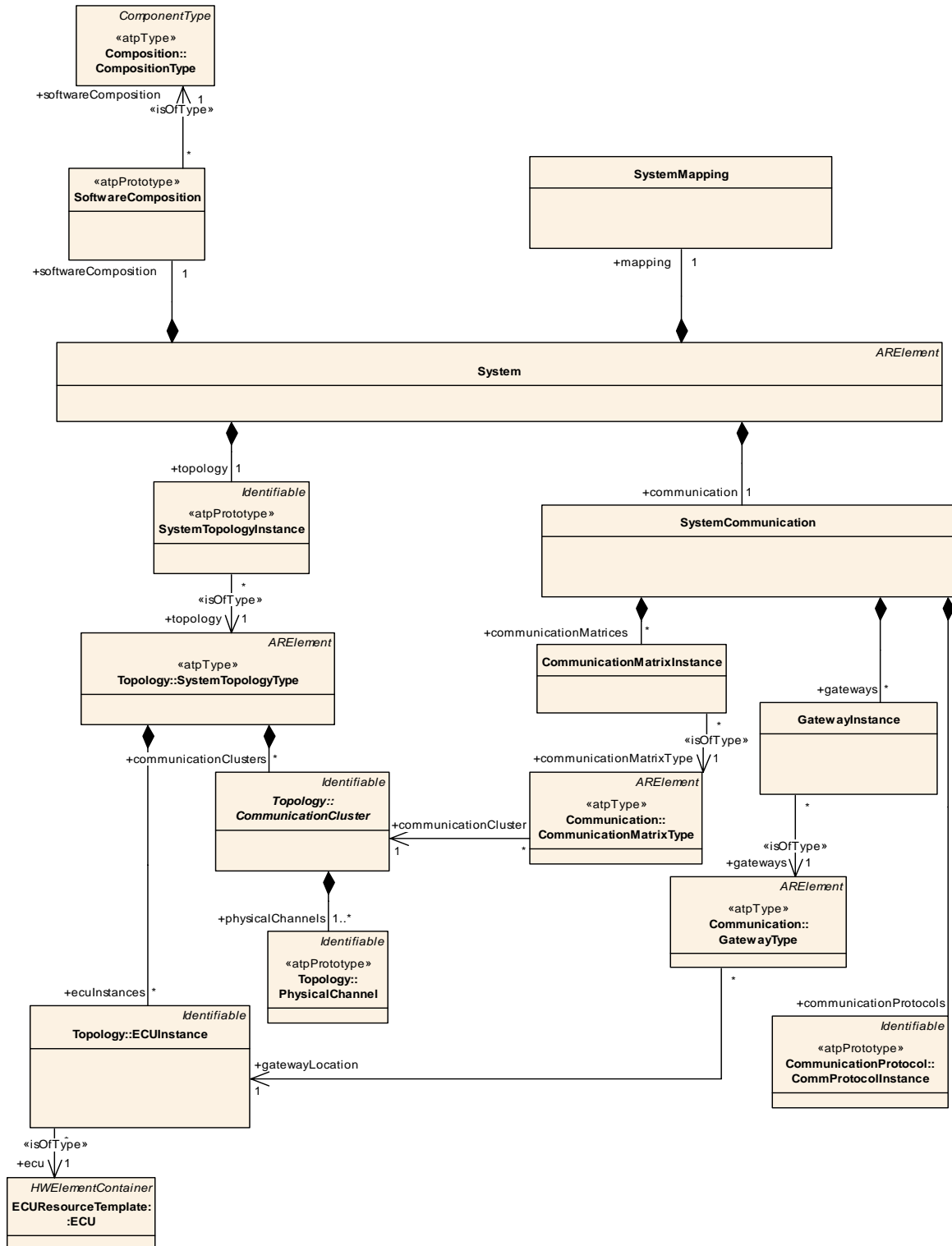


Figure 34 System model, top level classes

### 5.1.1. System (from SystemTemplate)

System is represented by the <<system>> stereotype which extends the Class metaclass (from Kernel).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b>        | <b>OCL Expression</b>   |
|---|-------------------------------|-----------------------------------|---|
| 1 | No Attributes or Operations   | ownedAttribute,<br>ownedOperation | self.ownedAttribute->size() = 0<br>and<br>self.ownedOperation->size() = 0 |

### 5.1.2. SystemTopologyInstance (from SystemTemplate)

SystemTopologyInstance is represented by the <<systemTopologyInstance>> stereotype which extends the Property metaclass (from Internal Structures).

This class is a PropertyEvaluator which can be represented by a UML Property.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>                           | <b>Constrained feature</b> | <b>OCL Expression</b>                     |
|---|---|----------------------------|---|
| 1 | Owned by a <<system>> class                             | Element::owner             | self.owner.oclsTypeOf(system)             |
| 2 | typed by a class with <<SystemTopologyType>> stereotype | TypedElement::type         | self.type.oclsTypeOf (SystemTopologyType) |

### 5.1.3. SystemCommunication (from SystemTemplate)

SystemCommunication is represented by the <<systemCommunication>> stereotype which extends the Class metaclass (from Kernel).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b>        | <b>OCL Expression</b>   |
|---|-------------------------------|-----------------------------------|---|
| 1 | Owned by <<system>> class     |                                   | Self.owner = <<system>> class   |
| 2 | No Attributes or Operations   | ownedAttribute,<br>ownedOperation | self.ownedAttribute->size() = 0<br>and<br>self.ownedOperation->size() = 0 |

### 5.1.4. CommunicationMatrixInstance (from SystemTemplate)

CommunicationMatrixInstance is represented by the <<communicationMatrixInstance>> stereotype which extends the Property metaclass (from Internal Structures).

This class is a PropertyEvaluator which can be represented by a UML Property.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>            | <b>Constrained feature</b> | <b>OCL Expression</b>                       |
|---|--|----------------------------|---|
| 1 | Owned by a <<systemCommunication>> class | Element::owner             | self.owner.ocllsTypeOf(systemCommunication) |

### 5.1.5. SystemMapping (from SystemTemplate)

SystemMapping is represented by the <<systemMapping>> stereotype which extends the Class metaclass (from Kernel).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b> | <b>OCL Expression</b>          |
|---|-------------------------------|----------------------------|--------------------------------|
| 1 | Owned by <<system>> class     | Element::owner             | self.owner.ocllsTypeOf(system) |

### 5.1.6. SoftwareComposition (from SystemTemplate)

SoftwareComposition is represented by the <<softwareComposition>> stereotype which extends the Property metaclass (from Internal Structures).

This class is a PropertyEvaluator which can be represented by a UML Property.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>                        | <b>Constrained feature</b> | <b>OCL Expression</b>                  |
|---|--|----------------------------|--|
| 1 | Owned by a <<system>> class                          | Element::owner             | self.owner.ocllsTypeOf(system)         |
| 2 | typed by a class with <<compositionType>> stereotype | TypedElement::type         | self.type.ocllsTypeOf(compositionType) |

## 5.2. SystemSignal

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.

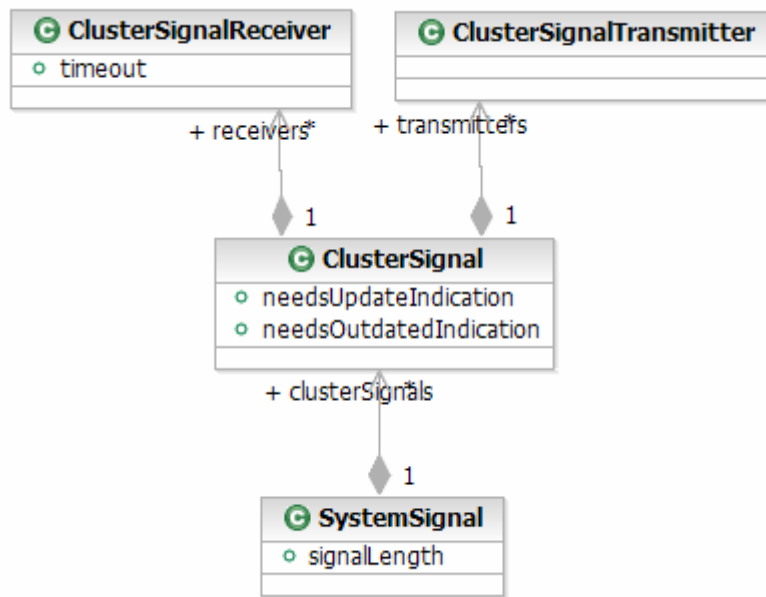


Figure 35 System Signals

### 5.2.1. SystemSignal (from SystemTemplate::SystemSignal)

SystemSignal is represented by the <<systemSignal>> stereotype which extends the Class metaclass (from kernel).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b>        | <b>OCL Expression</b>   |
|---|-------------------------------|-----------------------------------|---|
| 1 | No Attributes or Operations   | ownedAttribute,<br>ownedOperation | self.ownedAttribute->size() = 0<br>and<br>self.ownedOperation->size() = 0 |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>  | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|--------------|-------------|---------------------|
| 1 | Signal Length                | signalLength | Integer     | 1                   |

### 5.2.2. ClusterSignal (from SystemTemplate::SystemSignal)

ClusterSignal is represented by the <<clusterSignal>> stereotype which extends the Class metaclass (from kernel).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>   | <b>Constrained feature</b> | <b>OCL Expression</b>                |
|---|---------------------------------|----------------------------|--------------------------------------|
| 1 | Owned by <<systemSignal>> class | Element::owner             | self.owner.ocllsTypeOf(systemSignal) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>             | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------------------|-------------|---------------------|
| 1 | Update indication            | needsupdateindication   | Boolean     | 1                   |
| 2 | Outdated indication          | needsoutdatedindication | Boolean     | 1                   |

### 5.2.3. ClusterSignalReceiver (from SystemTemplate::SystemSignal)

ClusterSignalReceiver is represented by the <<clusterSignalReceiver>> stereotype which extends the Property metaclass (from Internal Structures).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>    | <b>Constrained feature</b> | <b>OCL Expression</b>                 |
|---|----------------------------------|----------------------------|---------------------------------------|
| 1 | Owned by <<clusterSignal>> class | Element::owner             | self.owner.ocllsTypeOf(clusterSignal) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 | Timeout                      | Timeout     | Integer     | 1                   |

### 5.2.4. ClusterSignalTransmitter (from SystemTemplate::SystemSignal)

ClusterSignalTransmitter is represented by the <<clusterSignalTransmitter>> stereotype which extends the Property metaclass (from Internal Structures).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>    | <b>Constrained feature</b>        | <b>OCL Expression</b>   |
|---|----------------------------------|-----------------------------------|---|
| 1 | Owned by <<clusterSignal>> class | Element::owner                    | self.owner.ocllsTypeOf(clusterSignal)                                     |
| 2 | No Attributes or Operations      | ownedAttribute,<br>ownedOperation | self.ownedAttribute->size() = 0<br>and<br>self.ownedOperation->size() = 0 |

### 5.3. FrameStructure

Generic model for communication frames on LIN, CAN and FlexRay. MOST is handled separately.

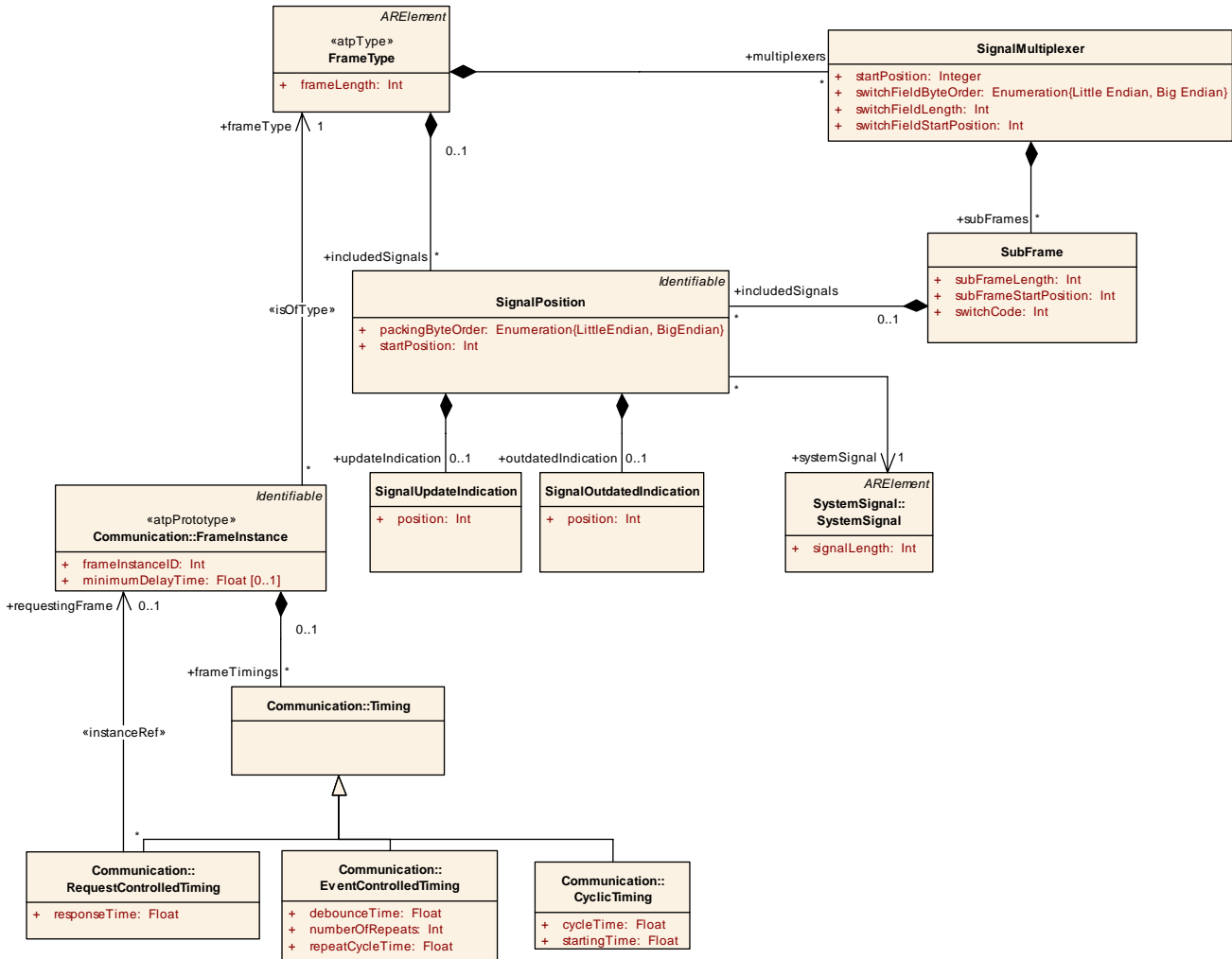


Figure 36 FrameType

#### 5.3.1. FrameType (from SystemTemplate::FrameStructure)

FrameType is represented by the <<frameType>> stereotype which extends the Class metaclass (from Kernel).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b>                                | <b>OCL Expression</b>   |
|---|-------------------------------|---|---|
| 1 | No Attributes or Operations   | Classifier::ownedAttribute,<br>Classifier::ownedOperation | self.ownedAttribute->size() = 0<br>and<br>self.ownedOperation->size() = 0 |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 | Frame length                 | framelength | Integer     | 1                   |

### 5.3.2. SignalMultiplexer (from SystemTemplate::FrameStructure)

SignalMultiplexer is represented by the <<signalMultiplexer>> stereotype which extends the Property metaclass (from Internal Structures).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b> | <b>OCL Expression</b>           |
|---|-------------------------------|----------------------------|---------------------------------|
| 1 | Ownded by a <<frameType>>     | Element::owner             | self.owner.oclTypeIs(FrameType) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>              | <b>Type</b>    | <b>Multiplicity</b> |
|---|------------------------------|--------------------------|----------------|---------------------|
| 1 |                              | startPosition            | Integer        | 1                   |
| 2 |                              | switchFieldStartPosition | Integer        | 1                   |
| 3 |                              | switchFieldLength        | Integer        | 1                   |
| 4 |                              | switchFieldByteOrder     | ByteOrder_Enum | 1                   |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Type</b>    | <b>Literals</b>         |
|---|--------------------------------|----------------|-------------------------|
| 1 |                                | ByteOrder_Enum | littleEndian, bigEndian |

### 5.3.3. SubFrame (from SystemTemplate::FrameStructure)

SubFrame is represented by the <<subFrame>> stereotype which extends the Property metaclass (from Internal Structures).

SubFrame is a property of SignalMultiplexer which is itself a Property. Properties can own other Properties as a qualifier.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>  | <b>Constrained feature</b> | <b>OCL Expression</b>                    |
|---|--------------------------------|----------------------------|--|
| 1 | Owned by <<signalMultiplexer>> | Element::owner             | self.owner.oclsTypeOf(signalMultiplexer) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>           | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-----------------------|-------------|---------------------|
| 1 |                              | switchCode            | Integer     | 1                   |
| 2 |                              | subFrameStartPosition | Integer     | 1                   |
| 3 |                              | subFrameLength        | Integer     | 1                   |

### 5.3.4. SignalPosition (from SystemTemplate::FrameStructure)

SignalPosition is represented by the <<signalPosition>> stereotype which extends the Property metaclass (from Internal Structures).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>                 | <b>Constrained feature</b> | <b>OCL Expression</b>   |
|---|---|----------------------------|---|
| 1 | Owned by either <<frameType>> or <<subFrame>> | Element::owner             | self.owner.oclsTypeOf(frameType)<br>or<br>self.owner.oclsTypeOf(subFrame) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>      | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|------------------|-------------|---------------------|
| 1 |                              | startPosition    | Integer     | 1                   |
| 2 |                              | packingByteOrder | Integer     | 1                   |

### 5.3.5. SignalOutdatedIndication (from SystemTemplate::FrameStructure)

SignalOutdatedIndication is represented by the <<signalOutdatedIndication>> stereotype which extends the Property metaclass (from Internal Structures).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b> | <b>OCL Expression</b>                 |
|---|-------------------------------|----------------------------|---------------------------------------|
| 1 | Owned by <<signalPosition>>   | Element::owner             | self.owner.oclsTypeOf(signalPosition) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 |                              | position    | Integer     | 1                   |

### 5.3.6. SignalUpdateIndication (from SystemTemplate::FrameStructure)

SignalUpdateIndication is represented by the <<signalUpdateIndication>> stereotype which extends the Property metaclass (from Internal Structures).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b> | <b>OCL Expression</b>                 |
|---|-------------------------------|----------------------------|---------------------------------------|
| 1 | Owned by <<signalPosition>>   | Element::owner             | self.owner.oclsTypeOf(signalPosition) |

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i> | <i>Type</i> | <i>Multiplicity</i> |
|---|------------------------------|-------------|-------------|---------------------|
| 1 |                              | position    | Integer     | 1                   |

## 5.4. DataMapping

### 5.4.1. DataMapping (from SystemTemplate::DataMapping)

DataMapping is represented by the <<dataMapping>> stereotype which extends the Property metaclass (from Internal Structures).

The following semantic constraints are defined for this stereotype.

|   | <i>Constraint Description</i> | <i>Constrained feature</i> | <i>OCL Expression</i>                 |
|---|-------------------------------|----------------------------|---------------------------------------|
| 1 | Owned by <<systemMapping>>    | Element::owner             | self.owner.ocllsTypeOf(systemMapping) |

### 5.4.2. SenderReceiverToUnspecifiedConnectionMapping (from SystemTemplate::DataMapping)

SenderReceiverToUnspecifiedConnectionMapping is represented by the <<SenderReceiverToUnspecifiedConnectionMapping>> stereotype which is a specialization of the <<dataMapping>> stereotype.

This class has associations to two other classes; these will be represented using attributes.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <i>Attribute Description</i>   | <i>Name</i>            | <i>Type</i> | <i>Multiplicity</i> |
|---|--|------------------------|-------------|---------------------|
| 1 | Port Interface data element. This string needs a fully qualified name. | dataElement            | String      | 1                   |
| 2 | Topology. This string needs a fully qualified name.                    | unspecifiedConnections | String      | 0..*                |

### 5.4.3. SenderReceiverToSignalMapping (from SystemTemplate::DataMapping)

SenderReceiverToSignalMapping is represented by the <<SenderReceiverToSignalMapping>> stereotype which is a specialization of the <<dataMapping>> stereotype.

This class has two associations, these will be represented as attributes. The <<instanceRef>> between this event and the DataElementPrototype will be modeled as a string attribute holding the qualified name.

There are no semantic constraints defined. The following attributes are required for this stereotype

|   | <b>Attribute Description</b>  | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|---|-------------|-------------|---------------------|
| 1 | Port Interface data <<instanceRef>> holds the port interface reference as a qualified name string in the form (/model/package/./package/Type.Property.Property) | dataElem    | String      | 1                   |
| 2 | System signal holds the fully qualified name in the form (/model/package/./package/Type)  | signal      | String      | 1..*                |

## 5.5. SystemMappingReference

SystemMappingReference is represented by the <<systemMappingReference>> stereotype which extends the Dependency metaclass.

This stereotype will be used to implement the <<instanceRef>> stereotype for SwCompToEcuMapping, providing a hierarchical aware dependency between itself and its three dependencies.

This stereotype will implement the mechanism proposed by SysML's (see [13]) <<Binding>> stereotype.

Dependency has two associations, supplier and client; the former will be a type encapsulating a hierarchy with either an Implementation, ComponentPrototype or ECUInstance at the leaf property; the latter will be a SwCompToEcuMapping.

To model the nesting a string attribute will be used which will hold a list of property names from the outside in, starting from the type of the supplier dependency.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>  | <b>Constrained feature</b> | <b>OCL Expression</b>                      |
|---|--|----------------------------|--|
| 1 | Originates from a SwCompToEcuMapping   | Dependency::client         | self.client.oclsTypeOf(SwCompToEcuMapping) |
| 2 | String attribute path's leaf property must be either a ComponentPrototype, ECUInstance or Implementation | Not defined                | Not defined.                               |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b>   | <b>Name</b> | <b>Type</b> | <b>Multiplicity</b> |
|---|--|-------------|-------------|---------------------|
| 1 | Nesting path from supplier down into the hierarchy in the form (property1.property2...); property1 is an ownedAttribute of the class referenced by supplier. | path        | String      | 1                   |

### 5.5.1. SwCompToEcuMapping (from SystemTemplate::SWmapping)

SwCompToEcuMapping is represented by the <<swCompToEcuMapping>> stereotype which extends the Class metaclass (from Internal Structures).

This class maintains relationships between an ECUInstance, a ComponentPrototype and an Implementation. Each is an <<instanceRef>> that will be modelled using the SystemMapping stereotype; constraints are defined to enforce the mapping.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>  | <b>Constrained feature</b>   | <b>OCL Expression</b>  |
|---|--|------------------------------|--|
| 1 | Owned by <<SWCompToECUMappingGroup>>                                       | Element::owner               | Self.owner.oclIsTypeOf(SWCompToECUMappingGroup)  |
| 2 | Must have 2, at most 3 dependencies with the <<systemMapping>> stereotype. | Dependency::clientDependency | self.clientDependency->select( d   d.oclIsTypeOf(systemMapping))->size() =2 or self.clientDependency->select( d   d.oclIsTypeOf(systemMapping))->size() =3 |

For Notation, see [10] 3.3.16.1.

## 5.6. Communication

### 5.6.1. CommunicationMatrixType (from SystemTemplate::Communication)

CommunicationMatrixType is represented by the <<communicationMatrixType>> stereotype which extends the Property metaclass (from Internal Structures).

This class has an association to a CommunicationCluster, this will be modelled as an string attribute holding its qualified name.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b> | <b>OCL Expression</b>           |
|---|-------------------------------|----------------------------|---------------------------------|
| 1 | No owned operations           | Artifact::ownedOperation   | self.ownedOperation->size() = 0 |
| 2 | No owned attributes           | Artifact::ownedAttribute   | self.ownedOperation->size() = 0 |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>          | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|----------------------|-------------|---------------------|
| 1 | Communication Cluster.       | communicationCluster | String      | 1                   |

### 5.6.2. LinSchedulingTable (from SystemTemplate::Communication)

LinSchedulingTable is represented by the <<LINSchedulingTable>> stereotype which extends the Property metaclass (from Internal Structures).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>          | <b>Constrained feature</b> | <b>OCL Expression</b>                           |
|---|--|----------------------------|---|
| 1 | Owned by a <<communicationMatrixType>> | Element::owner             | self.owner.oclIsTypeOf(communicationMatrixType) |

### 5.6.3. LinSchedulingTableEntry (from SystemTemplate::Communication)

LinSchedulingTableEntry is represented by the <<LinSchedulingTableEntry>> stereotype which extends the Property metaclass (from Internal Structures).

This class is contained by a LinSchedulingTable (itself a UML Property) as a qualifier.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>     | <b>Constrained feature</b> | <b>OCL Expression</b>                      |
|---|-----------------------------------|----------------------------|--|
| 1 | Owned by a <<LinSchedulingTable>> | Element::owner             | self.owner.ocllsTypeOf(LinSchedulingTable) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>      | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|------------------|-------------|---------------------|
| 1 |                              | delayToNextFrame | Float       | 1                   |

### 5.6.4. EventTriggeredFrameSlot (from SystemTemplate::Communication)

EventTriggeredFrameSlot is represented by the <<eventTriggeredFrameSlot>> stereotype which is a specialization of the <<LinSchedulingTableEntry>> stereotype.

This class has an association to a LinFrameInstance, this will be modelled as a string attribute which contains a qualified name.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b>   | <b>Name</b>           | <b>Type</b> | <b>Multiplicity</b> |
|---|--|-----------------------|-------------|---------------------|
| 1 |  | eventTriggeredFrameID | Integer     | 1                   |
| 2 | LinFrameInstance, string containing qualified name of this class in the form (/model/package/../../package/Type.Property.Property) | unconditionalFrame    | String      | 1                   |

### 5.6.5. UnconditionalFrameSlot (from SystemTemplate::Communication)

UnconditionalFrameSlot is represented by the <<unconditionalFrameSlot>> stereotype which is a specialization of the <<LinSchedulingTableEntry>> stereotype.

This class has an association to a LinFrameInstance, this will be modelled as a string attribute which contains a qualified name.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b>   | <b>Name</b>        | <b>Type</b> | <b>Multiplicity</b> |
|---|--|--------------------|-------------|---------------------|
| 1 | LinFrameInstance, string containing qualified name of this class in the form (/model/package/../../package/Type.Property.Property) | unconditionalFrame | String      | 1                   |

### 5.6.6. SporadicFrameSlot (from SystemTemplate::Communication)

SporadicFrameSlot is represented by the <<sporadicFrameSlot>> stereotype which is a specialization of the <<LINSchedulingTableEntry>> stereotype.

This class has an association to a LinFrameInstance, this will be modelled as a string attribute which contains a qualified name.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b>  | <b>Name</b>        | <b>Type</b> | <b>Multiplicity</b> |
|---|---|--------------------|-------------|---------------------|
| 1 | LinFrameInstance, string containing in the form (/model/package/../../package/Type.Property.Property) | unconditionalFrame | String      | 1                   |

### 5.6.7. FrameInstance (from SystemTemplate::Communication)

FrameInstance is represented by the <<frameInstance>> stereotype which extends the Property metaclass (from Internal Structures).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>          | <b>Constrained feature</b> | <b>OCL Expression</b>                          |
|---|--|----------------------------|--|
| 1 | Owned by a <<communicationMatrixType>> | Element::owner             | self.owner.oclsTypeOf(communicationMatrixType) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>      | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|------------------|-------------|---------------------|
| 1 |                              | frameInstanceID  | Integer     | 1                   |
| 2 |                              | minimumDelayTime | Float       | 0..1                |

### 5.6.8. LinFrameInstance (from SystemTemplate::Communication)

LinFrameInstance is represented by the <<LINFrameInstance>> stereotype which is a specialization of the <<frameInstance>> stereotype.

This class has an <<instanceRef>> to a SignalPosition, this will be modelled as a string attribute which contains a qualified name.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <i>Attribute Description</i>  | <i>Name</i>         | <i>Type</i> | <i>Multiplicity</i> |
|---|---|---------------------|-------------|---------------------|
| 1 | SignalPosition string; Contains a qualified name in the form (/model/package/./package/Type.Property.Property); | responseErrorSignal | String      | 1                   |

### 5.6.9. CanFrameInstance (from SystemTemplate::Communication)

CanFrameInstance is represented by the <<CANFrameInstance>> stereotype which is a specialization of the <<frameInstance>> stereotype

There are no semantic constraints defined.

### 5.6.10. FlexrayFrameInstance (from SystemTemplate::Communication)

FlexrayFrameInstance is represented by the <<flexrayFrameInstance>> stereotype which is a specialization of the <<frameInstance>> stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i>     | <i>Type</i>   | <i>Multiplicity</i> |
|---|------------------------------|-----------------|---------------|---------------------|
| 1 |                              | slotID          | Integer       | 1                   |
| 2 |                              | baseCycle       | Integer       | 1                   |
| 3 |                              | cycleRepetition | fr_frame_Enum | 1                   |

The following enumerations are required,

|   | <i>Enumeration Description</i> | <i>Type</i>   | <i>Literals</i>        |
|---|--------------------------------|---------------|------------------------|
| 1 |                                | fr_frame_Enum | 1, 2, 4, 8, 16, 32, 64 |

### 5.6.11. SystemEventTrigger (from SystemTemplate::Communication)

SystemEventTrigger is represented by the <<systemEventTrigger>> stereotype which extends the Property metaclass (from Internal Structures).

This class is contained by a Condition, hence it will be modelled as a UML Property of the Condition class.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>  | <b>Constrained feature</b> | <b>OCL Expression</b>  |
|---|--|----------------------------|--|
| 1 | Owned by a <<sendCondition>> , <<startCondition>> or <<stopCondition>> | Element::owner             | self.owner.oclIsTypeOf(sendCondition) or self.owner.oclIsTypeOf(startCondition) or self.owner.oclIsTypeOf(stopCondition) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>          | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|----------------------|-------------|---------------------|
| 1 |                              | systemEventCondition | String      | 1                   |

### 5.6.12. SystemStateTrigger (from SystemTemplate::Communication)

SystemStateTrigger is represented by the <<systemStateTrigger>> stereotype which extends the Property metaclass (from Internal Structures).

This class is contained by an ActiveCondition, hence it will be modelled as a UML Property of the ActiveCondition class.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>  | <b>Constrained feature</b> | <b>OCL Expression</b>                   |
|---|--------------------------------|----------------------------|---|
| 1 | Owned by a <<activeCondition>> | Element::owner             | self.owner.oclIsTypeOf(activeCondition) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>          | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|----------------------|-------------|---------------------|
| 1 |                              | systemStateCondition | String      | 1                   |

### 5.6.13. ActiveCondition (from SystemTemplate::Communication)

ActiveCondition is represented by the <<activeCondition>> stereotype which extends the Property metaclass (from Internal Structures).

This class is contained by a subclass of Timing; this relationship will be modelled as a Property-Class relationship.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>  | <b>Constrained feature</b> | <b>OCL Expression</b>   |
|---|--|----------------------------|---|
| 1 | Owned by <<requestControlTiming>> or <<eventControlledTiming>> or <<cyclicTiming>> | Element::owner             | self.owner.oclIsTypeOf(requestControlTiming) or self.owner.oclIsTypeOf(eventControlledTiming) or self.owner.oclIsTypeOf(cyclicTiming) |

#### 5.6.14. SendCondition (from SystemTemplate::Communication)

SendCondition is represented by the <<sendCondition>> stereotype which extends the Property metaclass (from Internal Structures).

This class is contained by a subclass of Timing; this relationship will be modelled as a Property-Class relationship.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>      | <b>Constrained feature</b> | <b>OCL Expression</b>                         |
|---|------------------------------------|----------------------------|---|
| 1 | Owned by <<eventControlledTiming>> | Element::owner             | self.owner.oclIsTypeOf(eventControlledTiming) |

#### 5.6.15. StopCondition (from SystemTemplate::Communication)

StopCondition is represented by the <<stopCondition>> stereotype which extends the Property metaclass (from Internal Structures).

This class is contained by a subclass of Timing; this relationship will be modelled as a Property-Class relationship.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b> | <b>OCL Expression</b>                |
|---|-------------------------------|----------------------------|--------------------------------------|
| 1 | Owned by <<stopCondition>>    | Element::owner             | self.owner.oclIsTypeOf(cyclicTiming) |

#### 5.6.16. StartCondition (from SystemTemplate::Communication)

StartCondition is represented by the <<startCondition>> stereotype which extends the Property metaclass (from Internal Structures).

This class is contained by a subclass of Timing; this relationship will be modelled as a Property-Class relationship.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b> | <b>OCL Expression</b>                |
|---|-------------------------------|----------------------------|--------------------------------------|
| 1 | Owned by <<cyclicTiming>>     | Element::owner             | self.owner.oclIsTypeOf(cyclicTiming) |

### 5.6.17. SignalTrigger (from SystemTemplate::Communication)

SignalTrigger is represented by the <<signalTrigger>> stereotype which extends the Property metaclass (from Internal Structures).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>   | <b>Constrained feature</b> | <b>OCL Expression</b>  |
|---|---|----------------------------|--|
| 1 | Owned by <<sendCondition>> or <<startCondition>> or <<stopCondition>> | Element::owner             | self.owner.oclIsTypeOf(sendCondition) or self.owner.oclIsTypeOf(startCondition) or self.owner.oclIsTypeOf(stopCondition) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>        | <b>Type</b>    | <b>Multiplicity</b> |
|---|------------------------------|--------------------|----------------|---------------------|
| 1 |                              | dataEventCondition | dataEvent_Enum |                     |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Type</b>    | <b>Literals</b>                                    |
|---|--------------------------------|----------------|--|
| 1 |                                | dataEvent_Enum | onChange, onUpdate, onNotDefault, onValueCondition |

### 5.6.18. Timing (from SystemTemplate::Communication)

Timing is represented by the <<timing>> stereotype which extends the Property metaclass (from Internal Structures). This class is abstract.

Timing is aggregated by either FrameInstance or ClusterSignal, this will be modelled as a Property-Class relationship.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>                   | <b>Constrained feature</b> | <b>OCL Expression</b>  |
|---|---|----------------------------|--|
| 1 | Owned by <<frameInstance>> or <<clusterSignal>> | Element::owner             | self.owner.oclIsTypeOf(frameInstance) or self.owner.oclIsTypeOf(clusterSignal) |

### 5.6.19. UnspecifiedTiming (from SystemTemplate::Communication)

UnspecifiedTiming is represented by the <<unspecifiedTiming>> stereotype which is a specialization of the <<timing>> stereotype.

There are no semantic constraints defined.

### 5.6.20. EventControlledTiming (from SystemTemplate::Communication)

EventControlledTiming is represented by the <<eventControlledTiming>> stereotype which is a specialization of the <<timing>> stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i>     | <i>Type</i> | <i>Multiplicity</i> |
|---|------------------------------|-----------------|-------------|---------------------|
| 1 |                              | debouceTime     | Float       | 1                   |
| 2 |                              | numberOfRepeats | Integer     | 1                   |
| 3 |                              | repeatCycleTime | Float       | 1                   |

### 5.6.21. CyclicTiming (from SystemTemplate::Communication)

CyclicTiming is represented by the <<cyclicTiming>> stereotype which is a specialization of the <<timing>> stereotype.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i>  | <i>Type</i> | <i>Multiplicity</i> |
|---|------------------------------|--------------|-------------|---------------------|
| 1 |                              | cycleTime    | Float       | 1                   |
| 2 |                              | startingTime | Float       | 1                   |

### 5.7. Topology

Defines which ECU's are present in a car, which buses are defined and how they're all are connected.

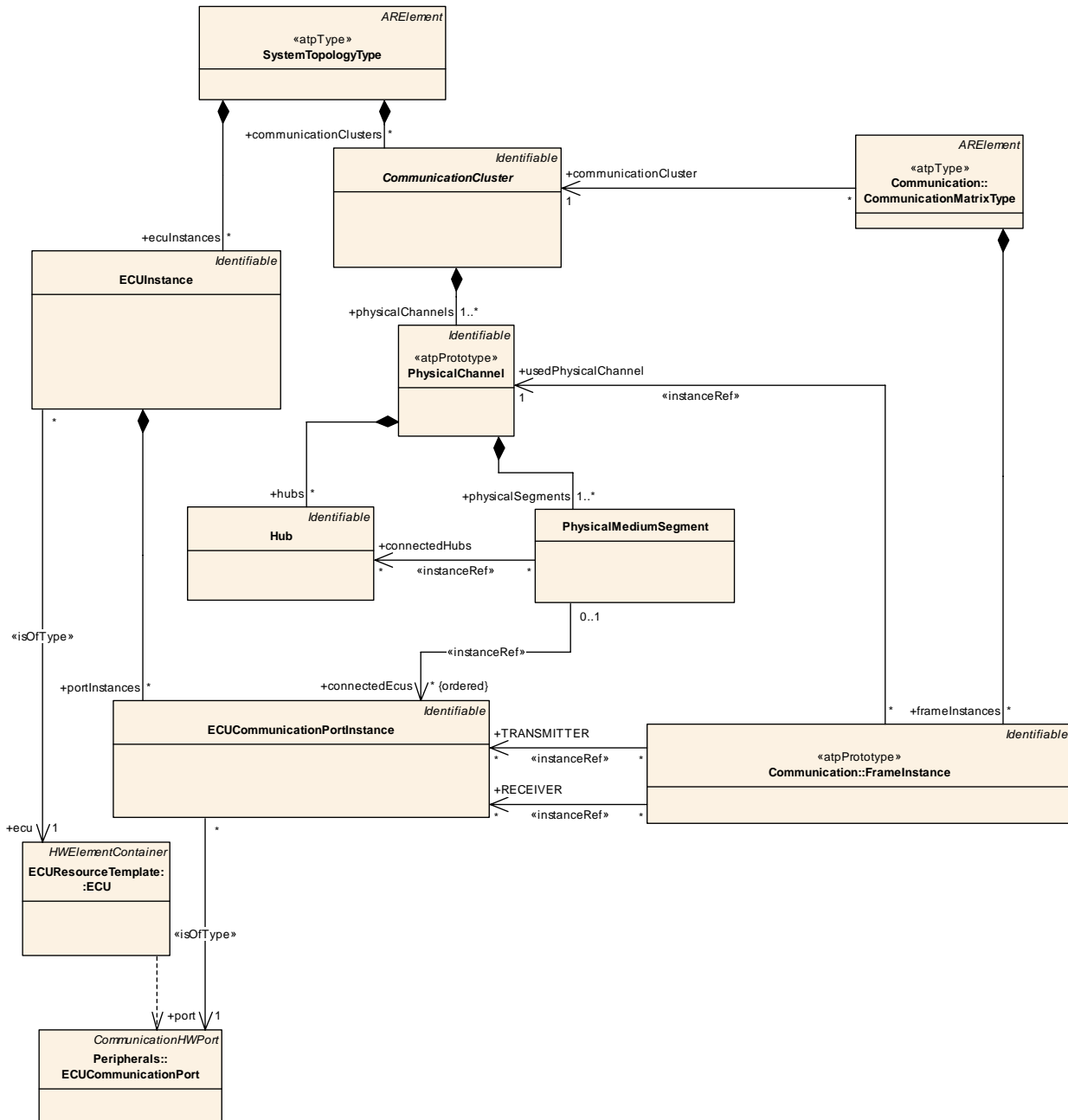


Figure 37 Topology

### 5.7.1. SystemTopologyType (from SystemTemplate::Topology)

SystemTopologyType is represented by the <<systemTopologyType>> stereotype which extends the Class metaclass (from Kernel).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b> | <b>Constrained feature</b>        | <b>OCL Expression</b>   |
|---|-------------------------------|-----------------------------------|---|
| 1 | No Attributes or Operations   | ownedAttribute,<br>ownedOperation | self.ownedAttribute->size() = 0<br>and<br>self.ownedOperation->size() = 0 |

### 5.7.2. PhysicalChannel (from SystemTemplate::Topology)

PhysicalChannel is represented by the <<physicalChannel>> stereotype which extends the Property metaclass (from Kernel).

To allow multiple network types to be modelled on the same topology diagram the physical channel will be modelled as a port owned by the specific communication instance (I.e. Flexray Cluster).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>                     | <b>Constrained feature</b> | <b>OCL Expression</b>                           |
|---|---|----------------------------|---|
| 1 | Must be owned by a <<communicationCluster>> class | Element::owner             | self.owner.oclsTypeOf(commu<br>nicationCluster) |

### 5.7.3. PhysicalMediumSegment (from SystemTemplate::Topology)

PhysicalMediumSegment is represented by the <<physicalMediumSegment>> stereotype which extends the Connector metaclass (from Internal Structures).

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>   | <b>Constrained feature</b> | <b>OCL Expression</b>  |
|---|---|----------------------------|--|
| 1 | Can only be connected to ports with the <<ECUCommunicationPortInstance>> and specializations stereotype OR class with <<hub>> stereotype. | ConnectorEnd::role         | self.end->forAll( e  <br>e.role.oclsKindOf(ECUComm<br>unicationPortInstance) or<br>e.role.oclsTypeOf(hub)) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>             | <b>Type</b>     | <b>Multiplicity</b> |
|---|------------------------------|-------------------------|-----------------|---------------------|
| 1 |                              | physicalMediumType      | mediumType_Enum | 1                   |
| 2 |                              | physicalMediumDataLines | Integer         | 1                   |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Type</b>     | <b>Literals</b>   |
|---|--------------------------------|-----------------|---|
| 1 |                                | mediumType_Enum | Optical, TwistedPair, ShieldedTwistedPair, Singlewire, Wireless |

#### 5.7.4. CommunicationCluster (from SystemTemplate::Topology)

CommunicationCluster is represented by the <<communicationCluster>> stereotype which extends the Property and Class metaclasses (from Kernel).

This class will appear as both a Property (part) and Class. PhysicalChannel's will appear as ports on the class.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>           | <b>Constrained feature</b> | <b>OCL Expression</b>                      |
|---|---|----------------------------|--|
| 1 | Owned by a <<systemTopologyType>> class | Element::owner             | self.owner.oclIsTypeOf(systemTopologyType) |

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>           | <b>Type</b> | <b>Multiplicity</b> |
|---|------------------------------|-----------------------|-------------|---------------------|
| 1 |                              | communicationSpeedBus | Integer     | 1                   |
| 2 |                              | protocolName          | String      | 0..1                |
| 3 |                              | protocolVersion       | String      | 0..1                |

#### 5.7.5. MOSTBus (from SystemTemplate::Topology)

MOSTBus is represented by the <<MOSTBus>> stereotype which is a specialization of the <<communicationCluster>> stereotype defined in Section 5.7.4.

There are no semantic constraints defined.

#### 5.7.6. FlexrayCluster (from SystemTemplate::Topology)

FlexrayCluster is represented by the <<flexrayCluster>> stereotype which is a specialization of the <<communicationCluster>> stereotype defined in Section 5.7.4.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|    | Attribute Description | Name                              | Type    | Multiplicity |
|----|-----------------------|-----------------------------------|---------|--------------|
| 1  |                       | wakeUpSymbolRxIdle                | Integer | 1            |
| 2  |                       | cycle                             | Float   | 1            |
| 3  |                       | minislotActionPointOffset         | Integer | 1            |
| 4  |                       | busGuardianEnablePart             | Integer | 1            |
| 5  |                       | macrotickDuration                 | Float   | 1            |
| 6  |                       | sampleClockPeriod                 | Float   | 1            |
| 7  |                       | networkManagementVectorLength     | Integer | 1            |
| 8  |                       | wakeUpSymbolTxIdle                | Integer | 1            |
| 9  |                       | dynamicSlotIdlePhase              | Integer | 1            |
| 10 |                       | staticSlotDuration                | Integer | 1            |
| 11 |                       | symbolWindow                      | Integer | 1            |
| 12 |                       | maxWithoutClockCorrectionFatal    | Integer | 1            |
| 13 |                       | transmissionStartSequenceDuration | Integer | 1            |
| 14 |                       | minislotDuration                  | Integer | 1            |
| 15 |                       | casRxLowMin                       | Integer | 1            |
| 16 |                       | syncNodeMax                       | Integer | 1            |
| 17 |                       | networkIdleTime                   | Integer | 1            |
| 18 |                       | listenNoise                       | Integer | 1            |
| 19 |                       | maxInitialisationError            | Float   | 1            |
| 20 |                       | casRxLowMax                       | Integer | 1            |
| 21 |                       | macroPerCycle                     | Integer | 1            |
| 22 |                       | numberOfStaticSlots               | Integer | 1            |
| 23 |                       | wakeUpSymbolRxWindow              | Integer | 1            |
| 24 |                       | wakeUpSymbolTxLow                 | Integer | 1            |
| 25 |                       | coldStartAttempts                 | Integer | 1            |
| 26 |                       | payloadLengthStatic               | Integer | 1            |
| 27 |                       | numberOfMinislots                 | Integer | 1            |
| 28 |                       | OffsetCorrectionStart             | Integer | 1            |
| 29 |                       | maxWithoutClockCorrectionPassive  | Integer | 1            |
| 30 |                       | wakeUpSymbolRxLow                 | Integer | 1            |
| 31 |                       | macroInitialOffset                | Integer | 1            |
| 32 |                       | actionPointOffset                 | Integer | 1            |

### 5.7.7. LINSubBus (from SystemTemplate::Topology)

LINSubBus is represented by the <<LINSubBus>> stereotype which is a specialization of the <<communicationCluster>> stereotype defined in Section 5.7.4.

There are no semantic constraints defined.

### 5.7.8. UnspecifiedConnection (from SystemTemplate::Topology)

UnspecifiedConnection is represented by the <<unspecifiedConnection>> stereotype which is a specialization of the <<communicationCluster>> stereotype defined in Section 5.7.4.

There are no semantic constraints defined.

### 5.7.9. CANBus (from SystemTemplate::Topology)

CANBus is represented by the <<CANBus>> stereotype which is a specialization of the <<communicationCluster>> stereotype defined in Section 5.7.4.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i>        | <i>Type</i>          | <i>Multiplicity</i> |
|---|------------------------------|--------------------|----------------------|---------------------|
| 1 |                              | canAddressingMode  | CAN_addressMode_Enum |                     |
| 2 |                              | samplingInstantMin | Integer              |                     |
| 3 |                              | samplingInstantMax | Integer              |                     |

The following enumeration is required,

|   | <i>Enumeration Description</i> | <i>Type</i>          | <i>Literals</i>    |
|---|--------------------------------|----------------------|--------------------|
| 1 |                                | CAN_addressMode_Enum | Standard, Extended |

### 5.7.10. ECUInstance (from SystemTemplate: Topology)

ECUInstance is represented by the <<ECUInstance>> stereotype which extends the Property and Class metaclasses (from InternalStructures)

AUTOSAR allows instances to add extra properties; in this case ports can be added to the instance. Although UML allows Properties to contain Properties (qualifiers) it was thought it would cause toolset implementation issues if a Property could contain a Port.

Instead an ECUInstance will be a part which is an instance of an automatically created subclass of ECU. This newly created subclass can own the ports.

The following semantic constraints are defined for this stereotype (Property).

|   | <i>Constraint Description</i>                   | <i>Constrained feature</i> | <i>OCL Expression</i>                      |
|---|---|----------------------------|--|
| 1 | Must be Owned by a <<systemTopologyType>> class | Element::owner             | self.owner.ocllsTypeOf(systemTopologyType) |
| 2 | Must reference an <<ecuInstance>>               | TypedElement::type         | self.type.ocllsKindOf(ecuInstance)         |

The following semantic constraints are defined for this stereotype (Class).

|   | <i>Constraint Description</i>                   | <i>Constrained feature</i> | <i>OCL Expression</i>   |
|---|---|----------------------------|---|
| 1 | Must be Owned by a <<systemTopologyType>> class | Element::owner             | self.owner.ocllsTypeOf(systemTopologyType)  |
| 2 | Must subclass <<ecu>>                           | Classifier::generalization | self.generalization.size()=1 and self.generalization->one(g   g.general.ocllsTypeOf(ecu)) |

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i>               | <i>Type</i> | <i>Multiplicity</i> |
|---|------------------------------|---------------------------|-------------|---------------------|
| 1 |                              | EcuAddress                | Integer     | 1                   |
| 2 |                              | powerSupply               | powerS_Enum | 1                   |
| 3 |                              | sleepModeSupported        | Boolean     | 1                   |
| 4 |                              | wakeUpOnInternalSupported | Boolean     | 1                   |
| 5 |                              | wakeUpOnIoSupported       | Boolean     | 1                   |

The following enumeration is required,

|   | <i>Enumeration Description</i> | <i>Type</i> | <i>Literals</i>                     |
|---|--------------------------------|-------------|-------------------------------------|
| 1 |                                | powerS_Enum | Battery, Ignition, Accessory, Other |

For Notation, see [10] 3.3.13.1.

### 5.7.11. ECUCommunicationPortInstance (from SystemTemplate::Topology)

ECUCommunicationPortInstance is represented by the abstract <<ECUCommunicationPortInstance>> stereotype which extends the Port metaclass (from Ports).

The following semantic constraints are defined for this stereotype.

|   | <i>Constraint Description</i>    | <i>Constrained feature</i> | <i>OCL Expression</i>              |
|---|----------------------------------|----------------------------|------------------------------------|
| 1 | Owned by a <<ecuInstance>> class | Element::owner             | self.owner.oclsTypeOf(ecuInstance) |

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i>               | <i>Type</i> | <i>Multiplicity</i> |
|---|------------------------------|---------------------------|-------------|---------------------|
| 1 |                              | commPortId                | Integer     | 1                   |
| 2 |                              | wakeUpOnCommPortSupported | Boolean     | 1                   |
| 3 |                              | communicationSpeedPort    | Integer     | 1                   |

For Notation, see [10] 3.3.14.1.

### 5.7.12. FlexrayCommunicationPortInstance (from SystemTemplate::Topology)

FlexrayCommunicationPortInstance is represented by the <<flexrayCommunicationPortInstance>> stereotype which is a specialization of the <<ECUCommunicationPortInstance>> stereotype defined in Section 5.7.11.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <i>Attribute Description</i> | <i>Name</i>          | <i>Type</i> | <i>Multiplicity</i> |
|---|------------------------------|----------------------|-------------|---------------------|
| 1 |                              | microPerCycle        | Integer     | 1                   |
| 2 |                              | dynamicSegmentEnable | Boolean     | 1                   |

|    |  |                             |         |   |
|----|--|-----------------------------|---------|---|
| 3  |  | maxDrift                    | Integer | 1 |
| 4  |  | externRateCorrection        | Integer | 1 |
| 5  |  | keySlotUsedForStartUp       | Boolean | 1 |
| 6  |  | delayCompensationB          | Integer | 1 |
| 7  |  | delayCompensationA          | Integer | 1 |
| 8  |  | keySlotUsedForSync          | Boolean | 1 |
| 9  |  | clusterDriftDamping         | Integer | 1 |
| 10 |  | MicroInitialOffsetB         | Integer | 1 |
| 11 |  | SingleSlotEnabled           | Boolean | 1 |
| 12 |  | acceptedStartupRange        | Integer | 1 |
| 13 |  | latestTX                    | Integer | 1 |
| 14 |  | microtickDuration           | Float   | 1 |
| 15 |  | MicroPerMacroNom            | Float   | 1 |
| 16 |  | listenTimeout               | Integer | 1 |
| 17 |  | allowHaltDueToClock         | Boolean | 1 |
| 18 |  | startUpNode                 | Boolean | 1 |
| 19 |  | syncSlot                    | Integer | 1 |
| 20 |  | allowPassiveToActive        | Integer | 1 |
| 21 |  | maximumDynamicPayloadLength | Integer | 1 |
| 22 |  | wakeUpPattern               | Integer | 1 |
| 23 |  | keySlotID                   | Integer | 1 |
| 24 |  | samplesPerMicrotick         | Integer | 1 |
| 25 |  | externOffsetCorrection      | Integer | 1 |
| 26 |  | MicroInitialOffsetA         | Integer | 1 |

### 5.7.13. CANCommunicationPortInstance (from SystemTemplate::Topology)

CANCommunicationPortInstance is represented by the <<CANCommunicationPortInstance>> stereotype which is a specialization of the <<ECUCommunicationPortInstance>> stereotype defined in Section 5.7.11.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|   | <b>Attribute Description</b> | <b>Name</b>                        | <b>Type</b>        | <b>Multiplicity</b> |
|---|------------------------------|------------------------------------|--------------------|---------------------|
| 1 |                              | receiveProcessingPeriod            | Float              | 1                   |
| 2 |                              | transmitProcessingPeriod           | Float              | 1                   |
| 3 |                              | protocolVersion                    | CAN_protoVers_Enum | 1                   |
| 4 |                              | maxPermittedProcessingPeriodJitter | Float              | 1                   |
| 5 |                              | portTermination                    | Boolean            | 1                   |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Type</b>        | <b>Literals</b> |
|---|--------------------------------|--------------------|-----------------|
| 1 |                                | CAN_protoVers_Enum | 2.0a, 2.0b      |

### 5.7.14. LINCommunicationPortInstance (from SystemTemplate::Topology)

LINCommunicationPortInstance is represented by the <<LINCommunicationPortInstance>> stereotype which is a specialization of the <<ECUCommunicationPortInstance>> stereotype defined in Section 5.7.11.

There are no semantic constraints defined.

The following attributes are required for this stereotype

|    | <b>Attribute Description</b> | <b>Name</b>             | <b>Type</b>        | <b>Multiplicity</b> |
|----|------------------------------|-------------------------|--------------------|---------------------|
| 1  |                              | p2Min                   | Integer            | 1                   |
| 2  |                              | supportSID              | String             | 1                   |
| 3  |                              | maxMessageLength        | Integer            | 1                   |
| 4  |                              | initialNAD              | Integer            | 1                   |
| 5  |                              | diagnosticAddress       | Integer            | 1                   |
| 6  |                              | protocolVersion         | LIN_protoVers_Enum | 1                   |
| 7  |                              | nodeRole                | linNode_Role_Enum  | 1                   |
| 8  |                              | supplierID              | Integer            | 1                   |
| 9  |                              | timeBaseJitter          | Float              | 1                   |
| 10 |                              | timeBase                | Float              | 1                   |
| 11 |                              | maxPermittedSlaveJitter | Float              | 1                   |
| 12 |                              | variantID               | Integer            | 1                   |
| 13 |                              | stMin                   | Float              | 1                   |
| 14 |                              | functionID              | Integer            | 1                   |

The following enumeration is required,

|   | <b>Enumeration Description</b> | <b>Type</b>        | <b>Literals</b> |
|---|--------------------------------|--------------------|-----------------|
| 1 |                                | linNode_Role_Enum  | Master, Slave   |
| 2 |                                | LIN_protoVers_Enum | 1.2,1.3,2.0     |

### 5.7.15. HUB (from SystemTemplate::Topology)

HUB is represented by the <<hub>> stereotype which extends the Property metaclass (from Kernel).

This stereotype will appear as a part on a structure diagram.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>                   | <b>Constrained feature</b> | <b>OCL Expression</b>                     |
|---|---|----------------------------|---|
| 1 | Must be Owned by a <<systemTopologyType>> class | Element::owner             | self.owner.oclsTypeOf(systemTopologyType) |
| 2 | Not typed                                       | TypedElement::type         | not self.type                             |

**5.7.16. Bus**

Bus is represented by the <<bus>> stereotype which extends the Property metaclass (from Kernel).

This class is not part of the AUTOSAR metamodel; however it will be required by some UML modelling tools to represent a communication bus if they don't support a connector with >2 endpoints.

The following semantic constraints are defined for this stereotype.

|   | <b>Constraint Description</b>                   | <b>Constrained feature</b> | <b>OCL Expression</b>                     |
|---|---|----------------------------|---|
| 1 | Must be Owned by a <<systemTopologyType>> class | Element::owner             | self.owner.ocIsTypeOf(systemTopologyType) |
| 2 | Not typed                                       | TypedElement::type         | not self.type                             |

## 6. References

### 6.1. UML2 Data Types

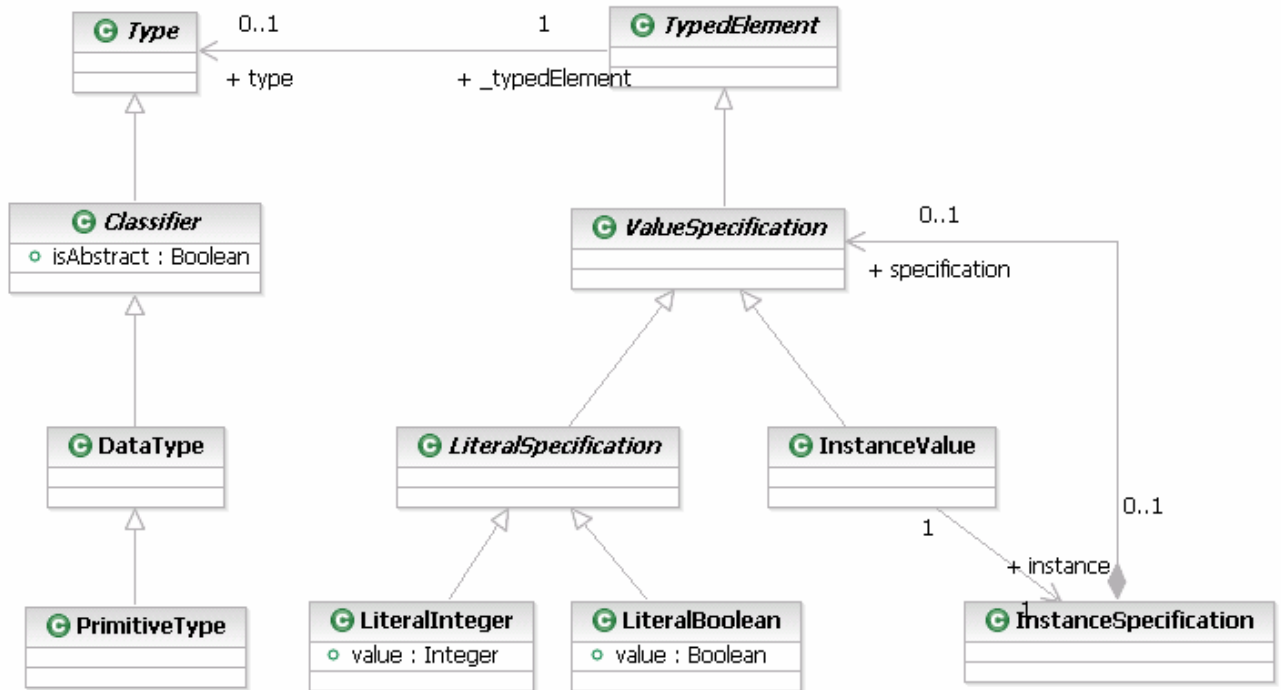


Figure 38: UML2 Data Types

### 6.2. Normative References to AUTOSAR documents

- [1] Glossary  
AUTOSAR\_Glossary.pdf
- [2] Requirements on Interoperability of Authoring Tools  
AUTOSAR\_RS\_InteroperabilityAuthoringTools.pdf
- [3] Methodology  
AUTOSAR\_Methodology.pdf
- [4] Software Component Template  
AUTOSAR\_SoftwareComponentTemplate.pdf
- [5] Specification of System Template  
AUTOSAR\_SystemTemplate.pdf
- [6] Specification of ECU Resource Template  
AUTOSAR\_ECUResourceTemplate.pdf
- [7] Metamodel  
AUTOSAR\_Metamodel.eap
- [8] Specification of Feature Definition of Authoring Tools  
AUTOSAR\_FeatureDefinition.pdf

- [9] Specification of Interaction with Behavioral Models  
AUTOSAR\_InteractionBehavioralModels.pdf
- [10] Specification of Graphical Notation  
AUTOSAR\_GraphicalNotification.pdf
- [11] Model Persistence Rules for XML  
AUTOSAR\_ModelPersistenceRulesXML.pdf
- [12] Template UML Profile and Modeling Guide  
AUTOSAR\_TemplateModelingGuide.pdf
- [13] Systems Modelling Language (SysML) Specification V1.0 alpha  
<http://www.sysml.org/artifacts/specs/SysMLv1.0a-051114R1.pdf>

### 6.3. Normative References to external documents

- [14] UML 2.0 Superstructure Specification  
<http://www.omg.org/technology/documents/formal/uml.htm>  
formal/05-07-04.pdf
- [15] Eclipse uml2 1.1 javadoc  
<http://download.eclipse.org/tools/uml2/1.1.0/javadoc/>
- [16] UML 2.0 OCL convenience document  
<http://www.omg.org/technology/documents/formal/uml.htm>  
ptc/05-06-06.pdf

## 6.4. Reference to Models

- [17] Eclipse uml2 profile  
AUTOSAR.profile.uml2
- [18] AUTOSAR UML 2.0 Profile for Rational Software Modeller  
Zip file containing an eclipse plugin which wraps the uml2 profile  
com.ibm.xtools.AUTOSAR.core\_1.0.0.zip
- [19] WiperWasher demo model, suitable for Rational Software Architect  
wiperWasher.emx
- [20] Rational Software Modeller download  
[www.ibm.com/software/rational/offerings/design.html](http://www.ibm.com/software/rational/offerings/design.html)