Document Title	Specification of Graphical Notation
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Identification No	206
Document Classification	Auxiliary

Document Version	1.0.5
Document Status	Final
Part of Release	3.1
Revision	0001

Document Change History

Date	Version	Changed by	Change Description
23.06.2008	1.0.5	AUTOSAR Administration	Legal Disclaimer revised
31.10.2007	1.0.4	AUTOSAR Administration	• Document meta information extended • Small layout adaptations made
24.01.2007	1.0.3	AUTOSAR Administration	• “Advice for users” revised • “Revision Information” added
04.12.2006	1.0.2	AUTOSAR Administration	Legal Disclaimer revised
26.06.2006	1.0.1	AUTOSAR Administration	Layout Adaptations
09.05.2006	1.0.0	AUTOSAR Administration	Initial release

Page left intentionally blank

Disclaimer

Advice to users of AUTOSAR Specification Documents:

Table of Contents

1 Introduction

The use of models is a well established approach to handle the complexity when engineering complex software. Various standards have been introduced in several domains to improve the interchangeability of models. In the field of automotive software components, models are already often created by using different, non-standardized and tool-specific graphical notations. There is demand to introduce standardization to these models by the AUTOSAR standard.

1.1 Aspects of AUTOSAR authoring tools (non-normative)

The AUTOSAR methodology document [3] describes the major steps of a development of system with AUTOSAR: from the system level to the generation of an ECU executable. It describes the dependencies of work-products and activities. Each authoring tool can support one or more activities.

The term *AUTOSAR authoring tool* refers to all tools that support the activities of interpretation, modification and creation of AUTOSAR models which describe a system as defined in the

- Software-Component Template [4]
- ECU Resource Template [6]
- and AUTOSAR System Template [5]

In particular, AUTOSAR authoring tools are required to be able to interpret, create or modify AUTOSAR XML descriptions (i.e. the XML representation of AUTOSAR models, see [1].

Figure 1-1 sketches the descriptions that can be maintained by AUTOSAR authoring tools within the AUTOSAR methodology (for a detailed description of the notation see [3]). According to the AUTOSAR methodology, the System Configuration Input consists of models describing software-components, ECU hardware and some system constraints.
The formal description of AUTOSAR software-components does not cover the behavior implementation. The latter is intentionally left to dedicated Behavior Modeling Tools (BMT). It is therefore necessary to bridge the gap between a software-component model and the corresponding behavior model created by a particular BMT. This task is carried out by the "Coupling Tool" mentioned in Figure 1-1.

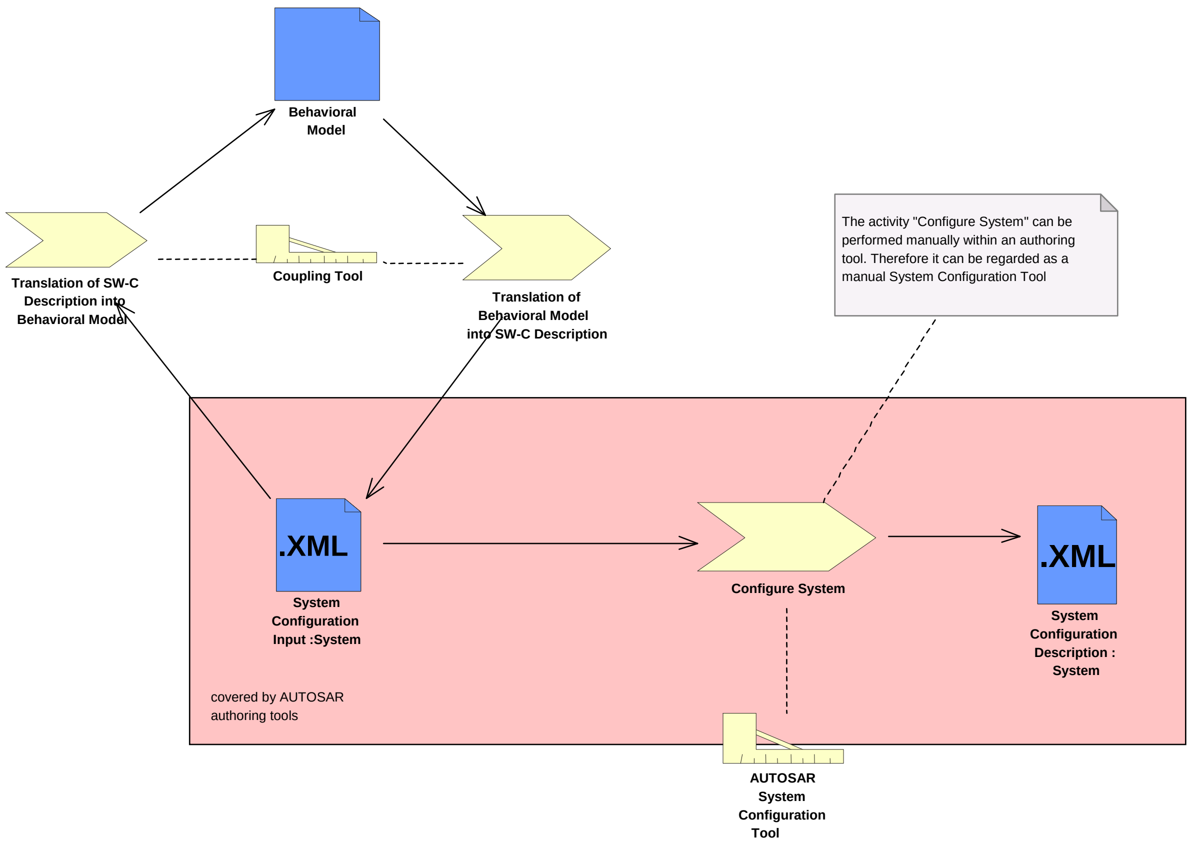

Behavioral Model
Translation of SW-C Description into Behavioral Model
Coupling Tool
Translation of Behavioral Model into SW-C Description
The activity "Configure System" can be performed manually within an authoring tool. Therefore it can be regarded as a manual System Configuration Tool
.XML
System Configuration Input :System
Configure System
.XML
System Configuration Description : System
covered by AUTOSAR authoring tools
AUTOSAR System Configuration Tool

Figure 1-1: Descriptions which can be created and modified by AUTOSAR authoring tools

The configuration of the System (as sketched in Figure 1-1) that produces the System Configuration Description as an output is a further aspect of AUTOSAR authoring tools. Please note that this task can be carried out manually or (to some extent) automatically. Since the automatic system configuration (although sketched as an "AUTOSAR System Configuration Tool") has not been addressed within AUTOSAR yet, the first approximation to AUTOSAR authoring tools as described in this document is focused on the manual creation of a System Configuration Description.

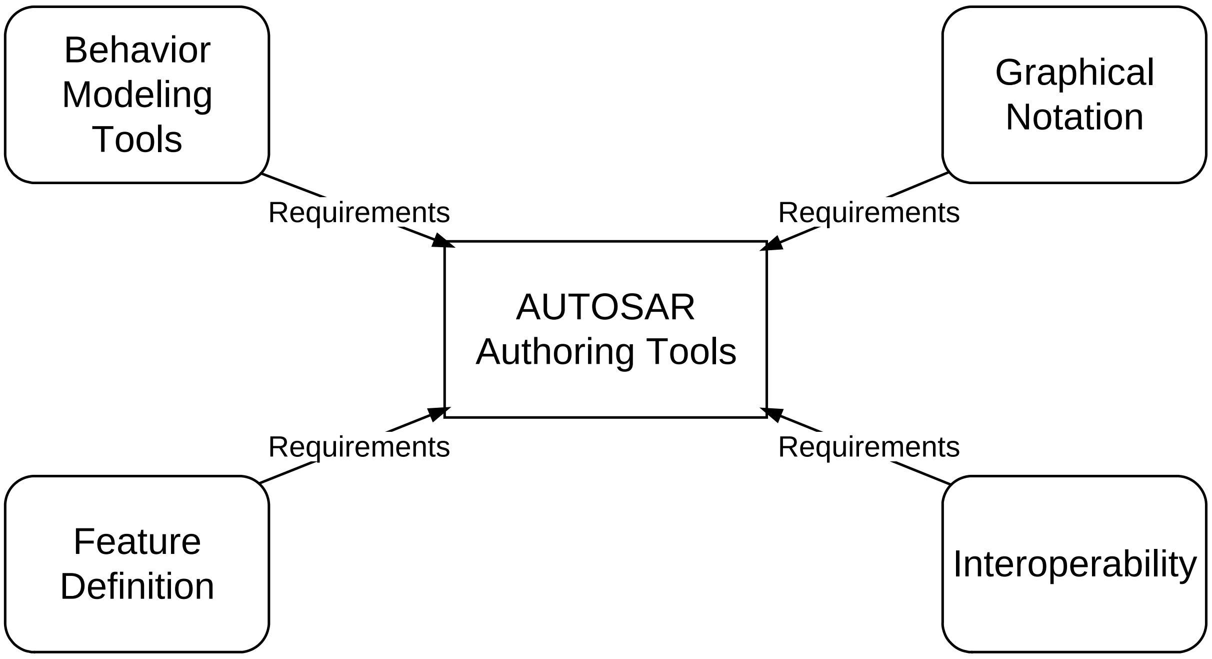


Figure 1-2: Aspects of AUTOSAR authoring tools

The description of AUTOSAR authoring tools covers several important aspects as depicted in Figure 1-2. Please note that the description of all these aspects results in the formulation of requirements on AUTOSAR authoring tools.
Each aspect as depicted in Figure 1-2 is described in a separate AUTOSAR document. In other words: beyond the scope of this document at hand, a separate discussion of specific aspects of AUTOSAR authoring tools (as depicted in Figure 1-2) is available (in a separate document for each aspect):

- **Specification of Feature Definition of Authoring Tools [8]**
The document gives a recommendation for a stepwise implementation of the overall AUTOSAR concept with respect to the interchange descriptions, namely the Software-Component Template, the ECU Resource Template and the System Template. As the basis for a first implementation, a subset (corresponding to the definition of features) of the AUTOSAR templates mentioned above for a first implementation of AUTOSAR authoring tools is defined.

- **Specification of Interoperability of Authoring Tools [10]**
This document emphasizes on issues that might come up when exchanging AUTOSAR models between different tools. After describing some basic concepts of data exchange this document sketches strategies on how these issues can be resolved. Requirements on AUTOSAR authoring tools for ensuring interoperability are defined.

- **Specification of Interaction with Behavioral Models [9]**
The document ”AUTOSAR Interaction with Behavioral Models” lists use-cases for behavior modeling within AUTOSAR. Parts of the AUTOSAR meta-model which are relevant for behavior modeling are identified. Requirements for the

interaction of software component descriptions with behavior models are derived.

- **Specification of Graphical Notation (this document)**
  This document defines the graphical AUTOSAR notation for AUTOSAR authoring tools. For example, the document provides a comprehensive schema for graphically modeling `CompositionTypes`. The graphical notation should be used as a guideline for implementing AUTOSAR authoring tools.

It is advised to read all of these documents in order to understand the overall concept of AUTOSAR authoring tools.
Please note that other tasks within the AUTOSAR concept, for example the creation of an ECU Configuration, are not in the scope of AUTOSAR authoring tools.

1.2 Origins and Goals

The AUTOSAR initiative has been established to standardize software infrastructure in automotive ECUs as well as corresponding information exchange formats between design tools.

The complete meta-model hierarchy for AUTOSAR templates is shown in Figure1-4. Unlike the classical four-layer architecture used by OMG, five metalevels are shown. Starting at the lowest, most concrete metalevel those are:

M0: AUTOSAR objects

This is the realization of an AUTOSAR system at work: real ECUs executing a software image containing for instance the windshield wiper control software.

M1: AUTOSAR models

Models on this metalevel are built by the AUTOSAR end-user (automotive engineers). They may define a software component called “windshield wiper” with a certain set of ports that is connected to another software component and so on. On this level all artifacts required to describe an AUTOSAR system are detailed, including re-usable types as well as specific instances.

An early idea within the AUTOSAR partnership was to use dedicated tools to directly work on the XML file representing the M1 models. It was then recognized that the use of graphical design tools increases usability and such tools should be used to create AUTOSAR models.

M2: AUTOSAR meta-model

On this metalevel the *vocabulary* is defined that later can be used by AUTOSAR end-users. E.g., it is defined that in AUTOSAR we have an entity called “software component” and another entity called “port”. The relation between those entities as well as their semantics is part of such an overall model.

M3: UML profile for AUTOSAR templates

The templates on M2 are built with the meta-model defined on M3. As discussed before this is UML plus a particular UML profile to better support template modeling work. Formally a template on M2 is still an instance of UML, but at the same time the template profile is *applied*, i.e. that additionally rules set out by the stereotypes in the profile need to be observed.

M4: Meta Object Facility

Just for completeness, OMG's MOF sits on the final metalevel M4. No further metalevels are required since MOF is designed to be reflective.

The data to be exchanged between the tools is defined formally by means of a meta-model [7], which has been created based on the AUTOSAR templates (eg. SWC template [4], ECU resource template [6]), as a model on the M2 level by means of UML. This defines the structure and semantic constraints of M1 models.

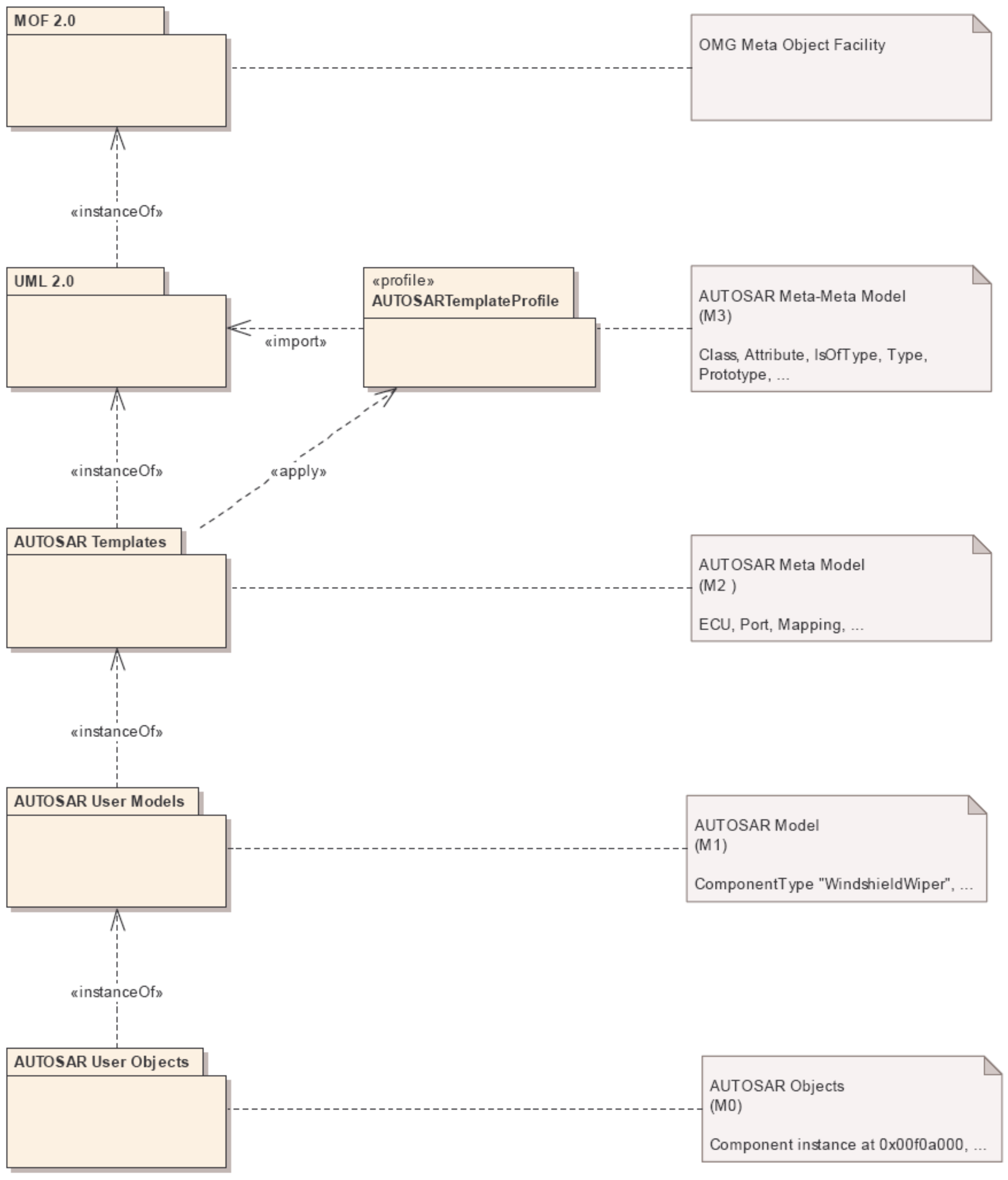


Figure 1-3: Meta-Model Hierarchy.

To define an actual AUTOSAR model, an M1 model has to be created. However, the M2 model does not imply a specific representation for M1 models. A graphical notation has to be agreed on.

The meta-model elements that are to be represented graphically are to be identified. Not all elements might be represented in a graphical notation. This probably has dependencies to the deliverable “Definition of a subset”, as a prioritization in that deliverable might affect the deliverable for graphical notation.

Since the M2 model is defined with UML [14] and UML is a standard within the software industry, the idea of using UML for M1 modeling comes to mind. The experience of using UML tools for the creation of AUTOSAR M1 models in prototypical projects suggest several difficulties that advise for the use of a dedicated AUTOSAR notation and toolset. When using modeling conventions and profiles to create M1 models for AUTOSAR by means of UML, all UML tools by default support as much of the UML standard as they can. However, this gives the user a wide choice of modeling elements that are not applicable in the AUTOSAR domain. Only very few tools allow the customization that would be necessary to constrict the possible choices to those relevant for the AUTOSAR domain.
Some AUTOSAR models cannot be expressed in a user-friendly manner by UML. While e.g. the communication channels between ECUs could be modeled by means of the deployment diagram’s communication channel, bus topologies are not easily represented in those diagrams.
While all those information could eventually be modeled using conventions, profiles (stereotypes, tagged values etc.), this would reduce the usability of the notation. In addition to the AUTOSAR semantic constraints, additional constraints would need to be defined, both formulated not referencing the original AUTOSAR M2, but in terms of UML M2.

1.3 Relation to other graphical notations within AUTOSAR

Diagrams and pictures to represent M1 models have already been used in AUTOSAR documents. Especially the notation used in the VFB document has been reused within other documents. The notation and symbols in those diagrams is taken into account for the definition of a graphical notation for tools. However, since the VFB notation was not designed for usability and tool support, several changes have to be made to allow for better user-friendliness within a tool.

2 Requirements Tracing

The requirements on this document are described in the document “Requirements on Graphical Notation“ [2]. Table 1 contains requirements trace matrix that indicates where these requirements are covered in this document.

Requirement	***Satisfied By***
ATREQ_019 Define diagrams	Section 3.4
ATREQ_020 Define a graphical notation	Section 3
ATREQ_023 Base Graphical on UML2	Section 3

Table 1: Requirements trace matrix

3 Specification of the notation

3.1 Strategies for the specification of the notation

3.1.1 Design criteria for symbols

To allow for usability, the set of graphical symbols should fulfill a few criteria:

- Simplicity: The symbols should be simple to be easily legible. They should not be made up from intricate graphical designs. As a general rule, a symbol should also be easily drawable by hand.
- Crispness: The symbol should evoke as much as possible in the reader the concept that it represents. E.g. a symbol representing a direction might have smiliarity to an arrow.
- Consistency: Similar concepts within the domain should be represented with similar model elements. E.g. the symbol for a port of a SWC should look similar to the symbol for the port of an ECU instance.

3.1.2 Relationship to UML and Domain Specific Modelling (DML)

The design of a graphical notation for a new domain is faced by the general decision of either reusing an existing standard notation or defining a new specific notation. This question has been discussed in the software engineering domain with respect to UML and Domain Specific Modelling (DML) [17]. Both approaches have its advantages and disadvantages. UML is a standard notation that is understood by a vast number of software engineers, however there are other notations that are in use especially in automotive settings. Using these notations is considered a forte of DML: “[DML] applies familiar terminology. There are no extra semantics: if you understand the domain you can understand the designs. This is unlike current methods where team members must first learn new semantics (e.g. UML) in addition to the domain semantics, and then learn how different developers make the mappings between them (and often how the semantics are twisted at one or both ends of the mapping).”[18]
It is expected that AUTOSAR models will be created both by users with an UML/SysML background (e.g. for the system architecture) and by users with a control engineering background. A notation should appeal to both groups.

3.2 Selection of elements to be graphically represented

The AUTOSAR meta-model spans several hundred elements. There is no benefit in representing every single element by a graphical notation. In several cases it is more user-friendly to display the parts of a composition by means of a textual or tabular notation.
Rules were followed in identifying model elements to be represented in a graphical notation:
Arguments for defining a symbol for a meta-model element:

- Established prior notations: Similar concepts in other domains that have been successfully represented in a graphical notation in their respective domains are candidates for a graphical representation in AUTOSAR (e.g. software architecture, topology)
- AUTOSAR concepts that are related to networks: Elements that appear in a context that is a kind of network of related elements are conveniently displayed in a graph and therefore in a graphical notation (e.g. software architecture)

Arguments against defining a symbol for a meta-model element:

- AUTOSAR elements that are parts in a composite relationship and do not have many references to other elements (e.g. `PropertyPrototype`)
- Concepts that are usually represented in non-graphic notations (e.g. `Assignment`)

3.3 Definition of graphical notation, mandatory notation and rep resentation options, Mapping of symbols to meta-model elements

The notation of the graphical elements is specified in this section. This includes the “mandatory” aspects, that have to be supported by all tool vendors and “representation options”, that specify alternative representations that can be used at the discretion of the tool vendors.

3.3.1 General Rules

This section specifies general rules that apply to all graphical elements that are discussed in this document.

3.3.1.1 Use of Color

The graphical representation must not rely on colors. It shall be able to represent all information in black and white. This is required both for printing and accessability by people with color blindness. Colors can be used as a specific option by user configuration. There are no normative guidelines on how to use color.

3.3.1.2 Screen Estate

The graphical representation implemented in tools should take as little screen estate as possible, thus allowing for compact diagrams that can show complex systems.

3.3.1.3 Labels

The labels should always be positioned for good readability. Most often that means horizontal alignment, although in some cases vertical alignment is justified, e.g. when labeling vertical connectors. If the containing element is to small to accommodate a

long label, the user or tool has to indicate that the name is not fully shown. A short hand notation can be used at own discretion (e.g. showing only a part and indicating the truncation with "…", scaling the text, providing a fly-over info box, breaking the label over various lines, etc.).
For AUTOSAR elements, the short-name is to be used for the name part of the label. The label can optionally use the fully qualified name for both the name and the type.

3.3.1.4 Zoom

The notation should be scalable so that it is readable on different levels of magnification.

3.3.1.5 Size

No specification about fixed sizes of the symbols are given. A tool might give the user the options to resize the symbols, automatically size symbols or provide a fixed size (e.g. for ports).

3.3.1.6 Element Attributes

Several AUTOSAR meta-model elements have attributes. If there is enough space, these attributes can be shown inside the graphical representation for the symbol in the textual format attribute name "=" value. Additionally and alternatively, a tool might implement other means of display (e.g. fly-over comments, dialog boxes).

3.3.1.7 Optional Elements

Variations on the graphical representation are defined in the sections "Presentation Options". These sections define variants or additional representation that can be supported by a tool, but need not be. If a tool choses to support that additional representation, it might support filters to allow the user to activate and deactivate these optional representations by means of filters (3.3.1.8).

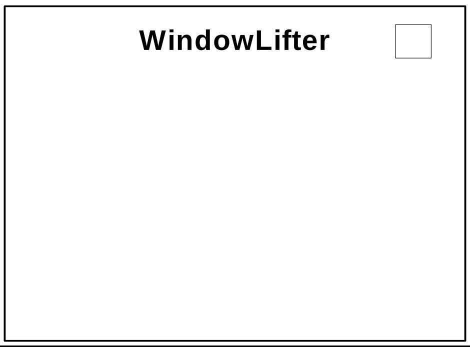
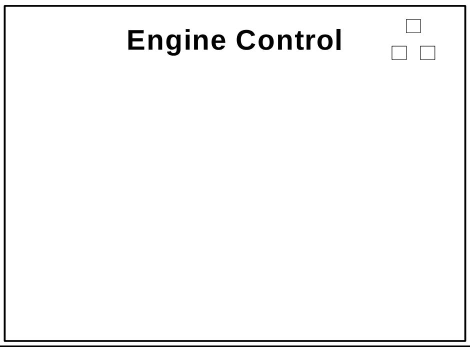
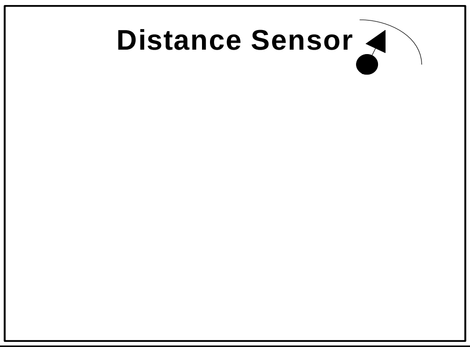
3.3.1.8 Filters

To improve the readability of complex diagrams, a tool may give the user the choice of filtering information (e.g. filtering certain kind of elements or by filtering optional elements as in 3.3.1.7).

3.3.2 Component Type

3.3.2.1 Notation

All `ComponentTypes` are represented by rectangles. The rectangle contains the component type's name and further graphical symbols to represent its concrete type (i.e. atomic, sensor-actuator,…). The name of the component must be placed anywhere inside the component.

Element	Graphical Notation	Comment
AtomicSoftware-Component Type		
CompositionType		
SensorActuator-SoftwareComponent-Type		

3.3.2.2 Presentation Options

A `CompositionType` is indicated by showing a symbol representing a subdiagram on the right side of the name compartment. It is up to the tools vendor to decide whether to show the contents of a `CompositionType` within the symbol itself or within a special diagram representing the contents.
A `SensorActuatorSoftwareComponentType` is indicated by an iconic measurement scale with a pointer.
Although AUTOSAR also has the notion of “`complex device drivers`”, these are currently not part of the meta-model as model elements of their own. A possible way to model them would be to use the `software-component` model elements and introduce a flag that specifies that the component is a `complex device driver`. However, this flag is not defined in the meta-model and cannot be exchanged by means of the exchange format. Any information about a component being a `complex device driver` would get lost in data exchange. In addition, as mentioned above from the interface point of view, a `complex device driver` looks just the

same as a regular software component. As a consequence, no dedicated symbol for complex device driver is introduced in this document. Should future versions of the exchange format support complex device drivers, a graphical icon could be introduced.

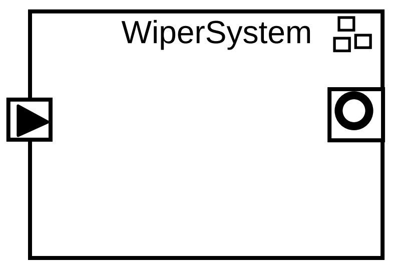
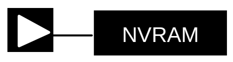
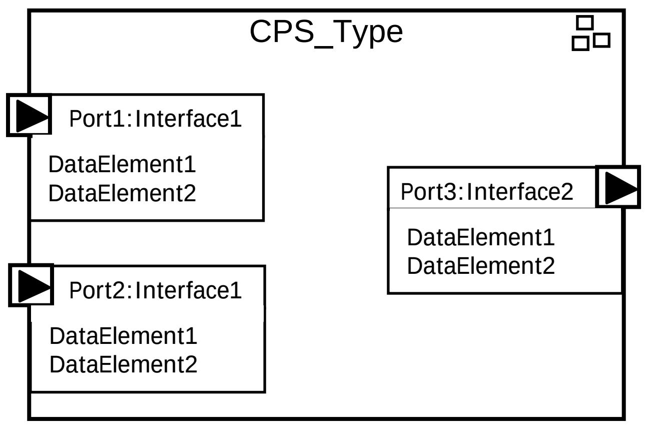
3.3.2.3 Rationale

Similar constructs are represented by similar representations in other standards, e.g. UML. These symbols will therefore be easily recognizable without the risk of confusing semantics of different notations.

3.3.3 Port

3.3.3.1 Notation

`PortPrototypes` are represented as rectangles that sit next to or on the border of the rectangle of a component type symbol. The contents of the port symbol consist of a graphical label defining the type of the port (`Require, Provide, Sender/Receiver, Client/Server`) and the name of the port. The tool must be able to display ports on any side of the component box. This means, tools must not restrict the location of the ports in the component based on their direction (`Provide/Require`). Additional information (the name of the port's associated interface and the interface's data items) is optional.

Element	Graphical Notation	Comment
Port symbol		Example of a require port with a SenderReceiverInterface and a ClientServerInterface showing the alternative positioning of the port symbol.
Port type adornments	▶ (pointing inward)	RPortPrototype with a SenderReceiverInterface
	◀ (pointing outward)	PPortPrototype with a SenderReceiverInterface
	○	PPortPrototype with a ClientServerInterface
	)	RPortPrototype with a ClientServerInterface
ServicePort		Exemplary for service port
Full port symbol	 CPS_Type Port1:Interface1 DataElement1 DataElement2 Port2:Interface1 DataElement1 DataElement2 Port3:Interface2 DataElement1 DataElement2	Two RPortPrototype and one PPortPrototype shown with the types of the associated interfaces and the data elements of the interfaces.

3.3.3.2 Presentation Options

Diagrams are expected to show different aspects of the overall model. In some cases, it is necessary to show details of the ports, while in other cases an overview of the entire system is of interest. In the second case, symbols should take less screen space. The port name, interface type and the `DataElementPrototypes` of the `PortInterface` need not be shown in these cases. A tool might show the information behind the port by means of a fly-over, pop-up etc.

A `PortPrototype` that is a `service port` (i.e. connects to a port of the AUTOSAR services) is represented by an inverted standard port symbol The symbol has an additional label giving some information about the service the port is connected to (e.g. NVRAM). `Service ports` have an implicit connection to the service that is shown in their label. They cannot be connected to other `PortPrototypes` on the same diagram.

The port symbols can also be drawn without filling the triangles, especially when drawing diagrams by hand.

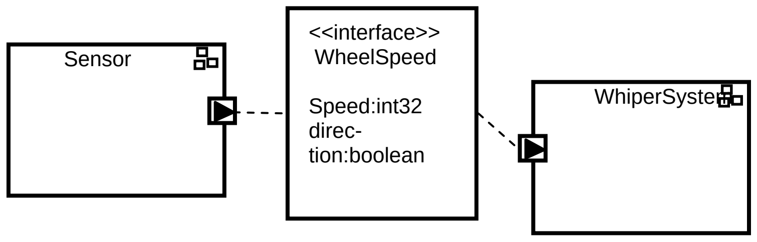
3.3.3.3 Rationale

Similar constructs are represented by similar representations in other standards, e.g. UML. These symbols will therefore be easily recognizable without the risk of confusing semantics of different notations. The "C" like symbol for a require port with a client interface is a mnemonic for "client". To distinguish elements in bad screen resolutions it is recommended to represent the `sender/receiver port` symbols solid, while `client/server` are hollow."

3.3.4 Interfaces

3.3.4.1 Notation

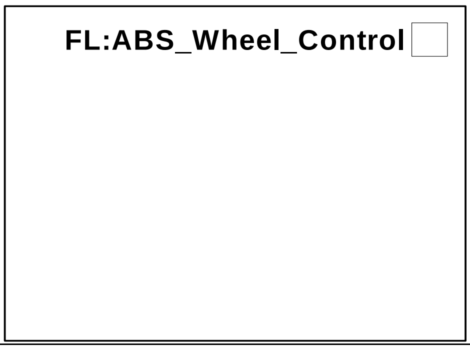
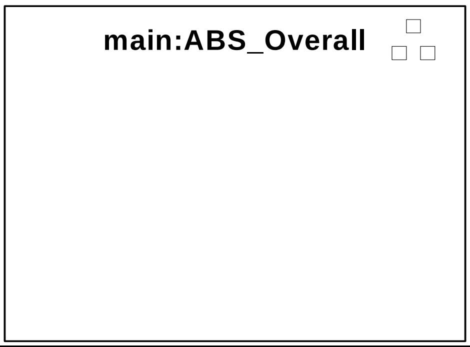
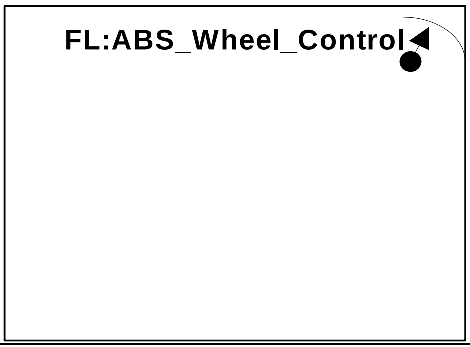
`PortInterfaces` are modelled as rectangles. The rectangle has a lable that shows the interface name. The interface's data items are listed textually within the rectangle. The assignment of an `PortInterface` to a `PortPrototype` is shown by a dashed line.

Element	Graphical Notation	Comment
PortInterface	 Sensor <<interface>> WheelSpeed Speed:int32 direc- tion:boolean WhiperSyste	

3.3.5 Component Prototype

3.3.5.1 Notation

A `ComponentPrototype` is represented by a rectangle with the element's label and an icon to represent it's the concrete subtype of its type. The label contains the `ComponentPrototype's` name and type in the syntax **Name ":" Type**. If no graphical indication is given, an `AtomicSoftwareComponentType` is assumed. The icon must be placed inside the component and must be attached to the string that specifies the name and type of the `ComponentPrototype`.The `PortPrototypes` for a `ComponentPrototype's PortPrototypes` are shown in the same way as for the `ComponentType`.

Element	Graphical Notation	Comment
Prototype for `AtomicSoftwareComponentType`		
Prototype for `CompositionType`		
`SensorActuatorSoftwareComponentType`		

3.3.5.2 Presentation Options

A `CompositionPrototype` is indicated by showing a symbol representing a sub-diagram on the right side of the lablel. It is up to the tools vendor to decide whether to show the contents of a `CompositionPrototype` within the symbol itself or within a special diagram representing the contents.

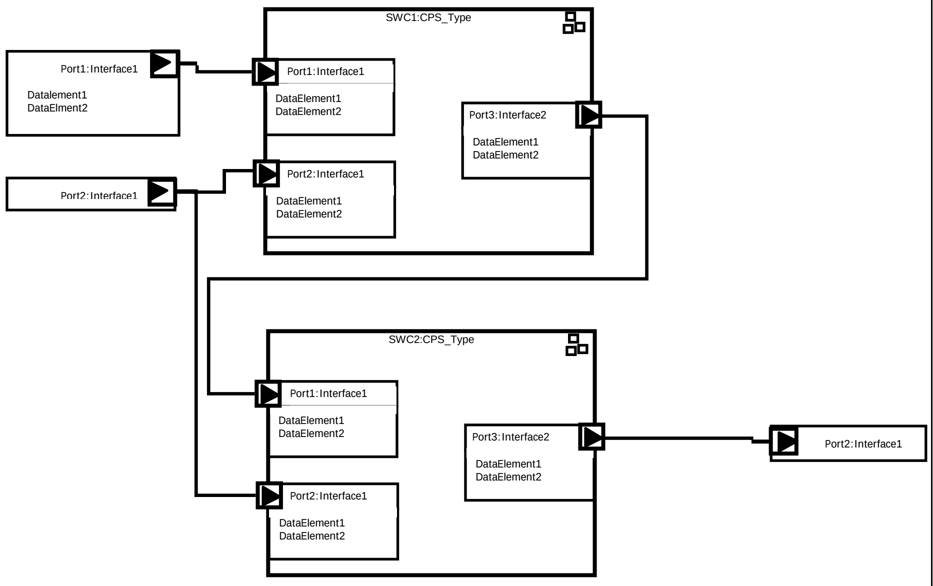
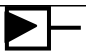
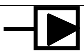
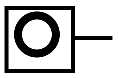
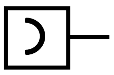
3.3.5.3 Rationale

Similar constructs are represented by similar representations in other standards, e.g. UML. These symbols will therefore be easily recognizable without the risk of confusing semantics of different notations.
The remarks made for `complex device` drivers in "3.3.2.3 Rationale" apply here as well.

3.3.6 Connector Prototype

3.3.6.1 Notation

A ConnectorPrototype is represented as a solid line between the symbols for its connected PortPrototypes. The inner view of a CompositionType is to be represented in a diagram of its own. In this diagram, the boundary of the CompositionType element is not displayed.
The lines representing ConnectorPrototypes can cross each other in the layout. Since a line crossing does not have any semantics, no special adornments are required for crossing lines in this case.

Element	Graphical Notation	Comment
DelegationConnectorPrototypes and AssemblyConnectorPrototypes.	 SWC1:CPS_Type Port1:Interface1 Datalement1 DataElment2 Port2:Interface1 Port1:Interface1 DataElement1 DataElement2 Port2:Interface1 DataElement1 DataElement2 Port3:Interface2 DataElement1 DataElement2 SWC2:CPS_Type Port1:Interface1 DataElement1 DataElement2 Port2:Interface1 DataElement1 DataElement2 Port3:Interface2 DataElement1 DataElement2 Port2:Interface1	Diagram showing 4 Delegation connectors (from the port representations on the left and right side) and a Assembly Connector (between the two software components)
Port Delegations		
		
		
		

3.3.6.2 Rationale

The same line type can be used since the actual type of ConnectorPrototype can be easily distinguished by looking if the connection is between ComponentPrototypes on the same hierarchy level (AssemblyConnectorPrototype) or between parent/child ComponentPrototypes (DelegationConnectorPrototype).

3.3.7 Internal Behavior

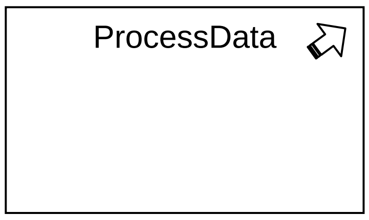
3.3.7.1 Notation

InternalBehavior in this context refers to the AUTOSAR-meta-model elements called "InternalBehavior". It does not describe the functional behavior itself, but only the static aspect of the behavior. It does not describe the functional behavior itself, but the scheduling relevant aspects of a component, i.e. the runnable entities and the events they respond to. Hence there is no explicit graphical symbol for Internal Behavior. It is shown implicitly via diagrams of RunnableEntities (Runnable Diagram).

3.3.8 Runnable Entity

3.3.8.1 Notation

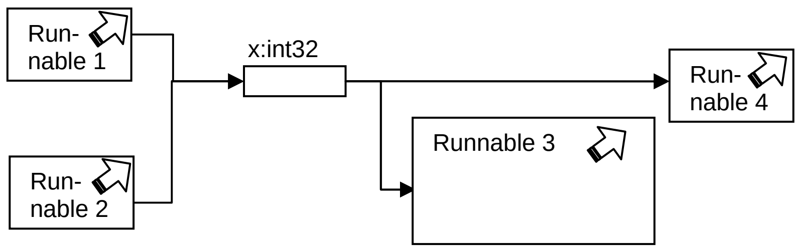
A RunnableEntity is represented as a rectangle with an arrow icon. The label of the symbol contains the name of the RunnableEntity and can be placed anywhere inside the box.

Element	Graphical Notation	Comment
RunnableEntity		

3.3.9 Inter Runnable Variables

3.3.9.1 Notation

An InterRunnableVariable is represented by a small rectangle. The name of the variable is placed outside and above the square as textual label. Read and write access from a RunnableEntity to the variable is indicated by arrows.

Element	Graphical Notation	Com-ment
`Interrunnable Variables`	 Run-nable 1 Run-nable 2 x:int32 Runnable 3 Run-nable 4	

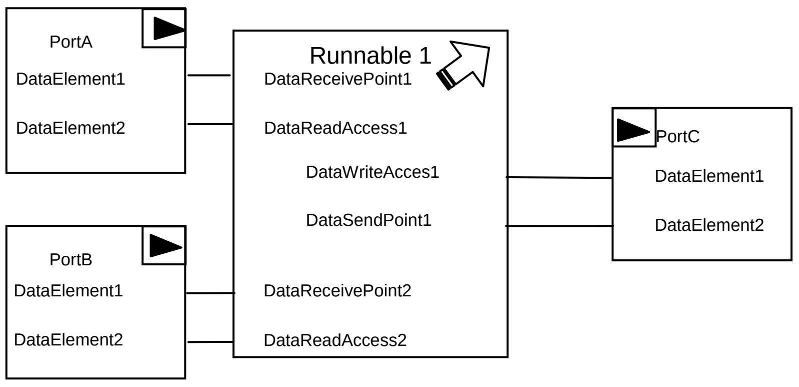
3.3.9.2 Presentation Options

The label of the variable can also be the name, followed by ":" and the type of the variable.

3.3.10 DataSendPoint, DataReceivePoint, DataWriteAccess, DataReadAccess

3.3.10.1 Notation

`DataSendPoint`, `DataReceivePoint`, `DataWriteAccess`, `DataReadAccess` and `ServerCallPoint` are all represented as textual labels showing the name of the element within the box for a `RunnableEntity`.
These elements can be connected to the data elements of a "full port representation" (cf. 3.3.3.2) by simple lines.

Element	Graphical Notation	Com-ment
Reference of `DataSendPoint`, `DataReceive-Point`, `DataWriteAcces` and `DataReadAc-cess` to `DataElement`	 PortA DataElement1 DataElement2 PortB DataElement1 DataElement2 Runnable 1 DataReceivePoint1 DataReadAccess1 DataWriteAcces1 DataSendPoint1 DataReceivePoint2 DataReadAccess2 PortC DataElement1 DataElement2	

3.3.10.2 Presentation Options

The label for the name of the element can be optionally followed by ":" and its meta-model element type, i.e. one of the str ings "`DataSendPoint`", "`DataReceive-Point`", "`DataWriteAcces`" , "`DataReadAccess`" or "`ServerCallPoint`"

3.3.10.3 Rationale

The connecting line need not indicate direction, since this can be derived from the port and the type of the element in the `RunnableEntity`.

3.3.11 RTEEvents

3.3.11.1 Notation

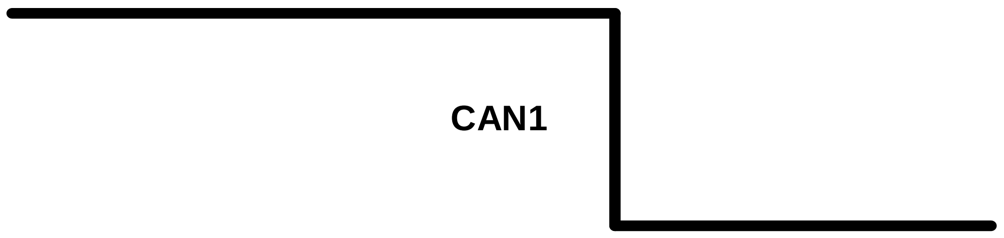
As `RTEEvents` are always connected to at most one `RunnableEntity`, they can be shown at the entity that they are associated with. Events that are not associated with any `RunnableEntity` are not shown. `RTEEvents` are specified as one or more textual symbols after the `RunnableEntity's` name, separated by ",".

- Timing Event : **"cyclic:" period**

3.3.12 CommunicationCluster

3.3.12.1 Notation

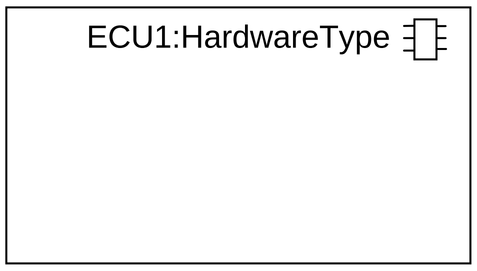
A `CommunicationCluster` is represented by a thick line. The line can have several corners to make use of diagram space.

Element	Graphical Notation
`Communication Cluster`	

3.3.13 ECUInstance

3.3.13.1 Notation

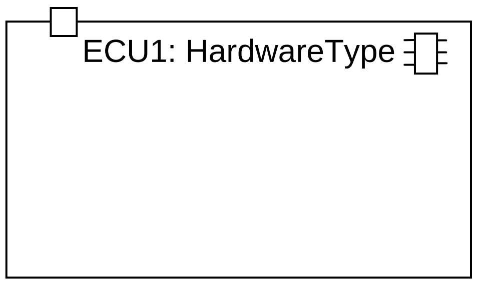
An `ECUInstance` is represented by a rectangle. The rectangle contains ECU's name and type and an icon to identify it as the symbol for an ECU. The name of the component must be placed anywhere inside the component. The remaining space of the inside of the rectangle my be used to show the deployed `software components`.

Element	Graphical Notation	Comment
ECUInstance		

3.3.14 ECUPortInstance

3.3.14.1 Notation

ECUPortInstance are represented as rectangles that sit next to or on the border of a component type symbol.

Element	Graphical Notation	Com-ment
ECUPortInstance		

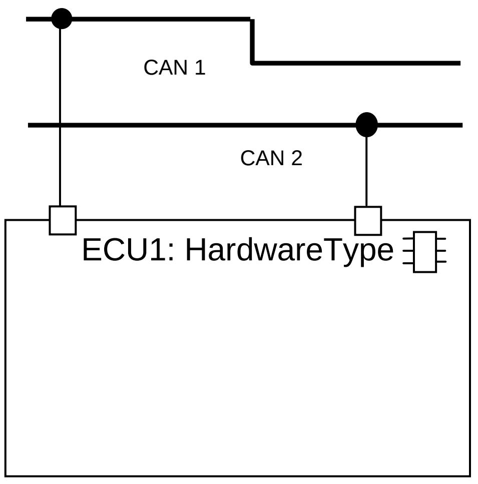
3.3.14.2 Presentation Options

The considerations for ports for ComponentTypes apply to this kind of ports.

3.3.15 Connection to Bus

3.3.15.1 Notation

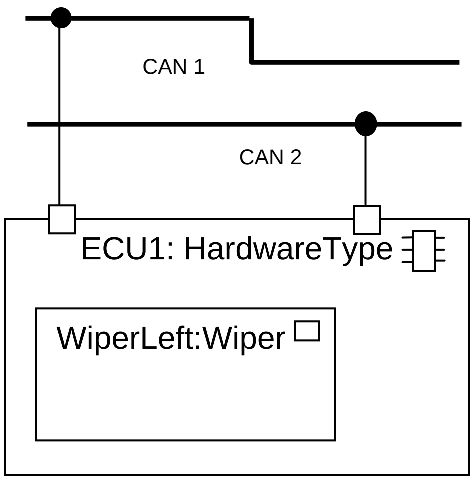
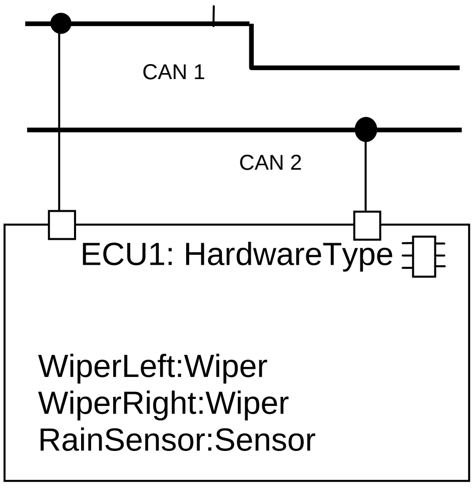
A `connection to bus` is represented as a solid line between the symbols for its connected `port` and the line of the `bus`.
The lines representing `connectors` can cross each other in the layout. To improve readability in complex diagrams, the connection to bus is displayed by a filled circle. Line crossings without this circle imply purely graphical crossing without semantics.

Element	Graphical Notation	Com-ment
`Connection to Bus`		

3.3.16 SwCompToEcuMapping

3.3.16.1 Notation

The mapping of a `ComponentProtoype` to an `ECUInstance` is shown by putting the graphical representation (3.3.5) for the `ComponentPrototype` that is to be mapped inside the symbol of the `ECU instance`.

Element	Graphical Notation	Com-ment
`SwCompToEcuMapping`		
Optional representation of mapped `Compo-nentPrototypes` in textual list form	 CAN 1 CAN 2 ECU1: HardwareType WiperLeft:Wiper WiperRight:Wiper RainSensor:Sensor	

3.3.16.2 Presentation Options

The box for the `ComponentPrototype` takes some screen estate. The tool might optionally show the mapped `ComponentPrototypes` as a vertical list of labels consisting of the name,":" and type of the `ComponentPrototype`.

If the tool displays the `ComponentPrototypes` in a graphical way, it can optionally show the `ports` and `connectors` of these `ComponentPrototypes`.

3.3.16.3 Rationale

A graphical representation of the `SwCompToEcuMapping` can by helpful for gaining an understanding of the system that is modelled. It might however be very complex when the system has a very fine granular mapping of a large number of software components.

3.4 Diagram taxonomy

The graphical elements will appear on diagrams. Different diagrams might be used to show different aspects of the modeled system. It should be specified which graphical elements can be used in which kind of diagrams.
It is expected that some tools might focus on specific aspects on AUTOSAR and provide editing capabilities for that aspect only, e.g. only the software-part and not the hardware-part. So it is not expected that all tools will implement all diagrams. If a tool implements a type of diagram with all the mandatory elements as defined in this document, it is considered to be "graphical notation compliant" for that type of diagram. The following table shows the diagram types and the graphical elements that are mandatory ("X") or optional ("O") in the diagrams.

	SWC and Interfaces Diagram	Composition Diagram	Topology and Mapping Diagram	Runnable Diagram
`Component Type`	X			
`Port`	X	X		
`Interfaces`	X			
`Component Prototype`		X	O	
`Connector Prototype`		X		
`Runnable Entity`				X
`Inter Runnable Variables`				X
`DataSendPoint, DataReceivePoint, DataWriteAccess, Data ReadAccess`				X
`RTEEvents`				X
CommunicationCluster			X	
`ECUInstance`			X	
`ECUPortInstance`			X	
`Connection to Bus`			X	

3.4.1 SWC and Interfaces

This diagram shows the association of `ports` to `interfaces`. However, since `ports` can be associated to interface by means of the naming (i.e. "port-name:interface"), such a diagram is not absolutely necessary for the specification.

However, from that diagram, a possible instantiation and a possible information flow could be derived.

3.4.2 Composition Diagram

The composition diagram shows the `composition/assembly` of various `software component prototypes` by connecting the `ports` of the `prototypes`. It is used to model an actual system, defining the instances of the `software component types` and the actual connections. By showing the internal structure of a `composite type` this diagram type can be used to show the system at different hierarchy levels, including the top level system view, since that is represented by the `composite` that types “`SoftwareComposition`” in the AUTOSAR meta-model.

3.4.3 Topology and Mapping Diagram

The topology diagram shows the buses and ECUs of a connected system. The support of showing the mapping (assignment of `software components` to ECUs) is optional, since a diagram with mapping can become very complex. A tool can choose to support a different representation, e.g. a tabular representation, for the mapping.

3.4.4 Runnable Diagram

The Runnable Diagram shows the internal structure of a `software component`.

4 References

4.1 Normative References to AUTOSAR documents

[1] Glossary
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_Glossary.pdf

[2] Requirements on Graphical Notation
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_RS_GraphicalNotation.pdf

[3] Methodology
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_Methodology.pdf

[4] Software Component Template
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_SoftwareComponentTemplate.pdf

[5] Specification of System Template
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_SystemTemplate.pdf

[6] Specification of ECU Resource Template
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_ECU_ResourceTemplate.pdf

[7] Meta-Model
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_MetaModel.EAP

[8] Specification of Feature Definition of Authoring Tools
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_FeatureDefinition.pdf

[9] Specification of Interaction with Behavioral Models
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_InteractionBehavioralModels.pdf

[10] Specification of Interoperability of Authoring Tools
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_InteroperabilityAuthoringTools.pdf

[11] Model Persistence Rules for XML
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_ModelPersistenceRulesXML.pdf

[12] Template UML Profile and Modeling Guide
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR TemplateModelingGuide.pdf,

4.2 Normative References to external documents

[13] XML Metadata Interchange (XMI) Specification version 2.0
http://www.omg.org/technology/documents/formal/xmi.htm

[14] UML 2.0 Superstructure Available Specification
http://www.omg.org/technology/documents/formal/uml.htm

[15] UML 2.0 Diagram Interchange final adopted specification
http://www.omg.org/cgi-bin/doc?ptc/2003-09-01

[16] Extensible Markup Language (XML) 1.1
http://www.w3.org/TR/xml11

4.3 Other References

[17] Wikipedia, *Domain-specific modelling*
http://en.wikipedia.org/wiki/Domain-specific_modelling

[18] Juha-Pekka Tolvanen & Steven Kelly, *Domain-specific modelling - 10 times faster*
http://www.esemagazine.co.uk/common/viewer/archive/2002/May/14/feature5.phtm