AUTOSAR

What is AUTOSAR?

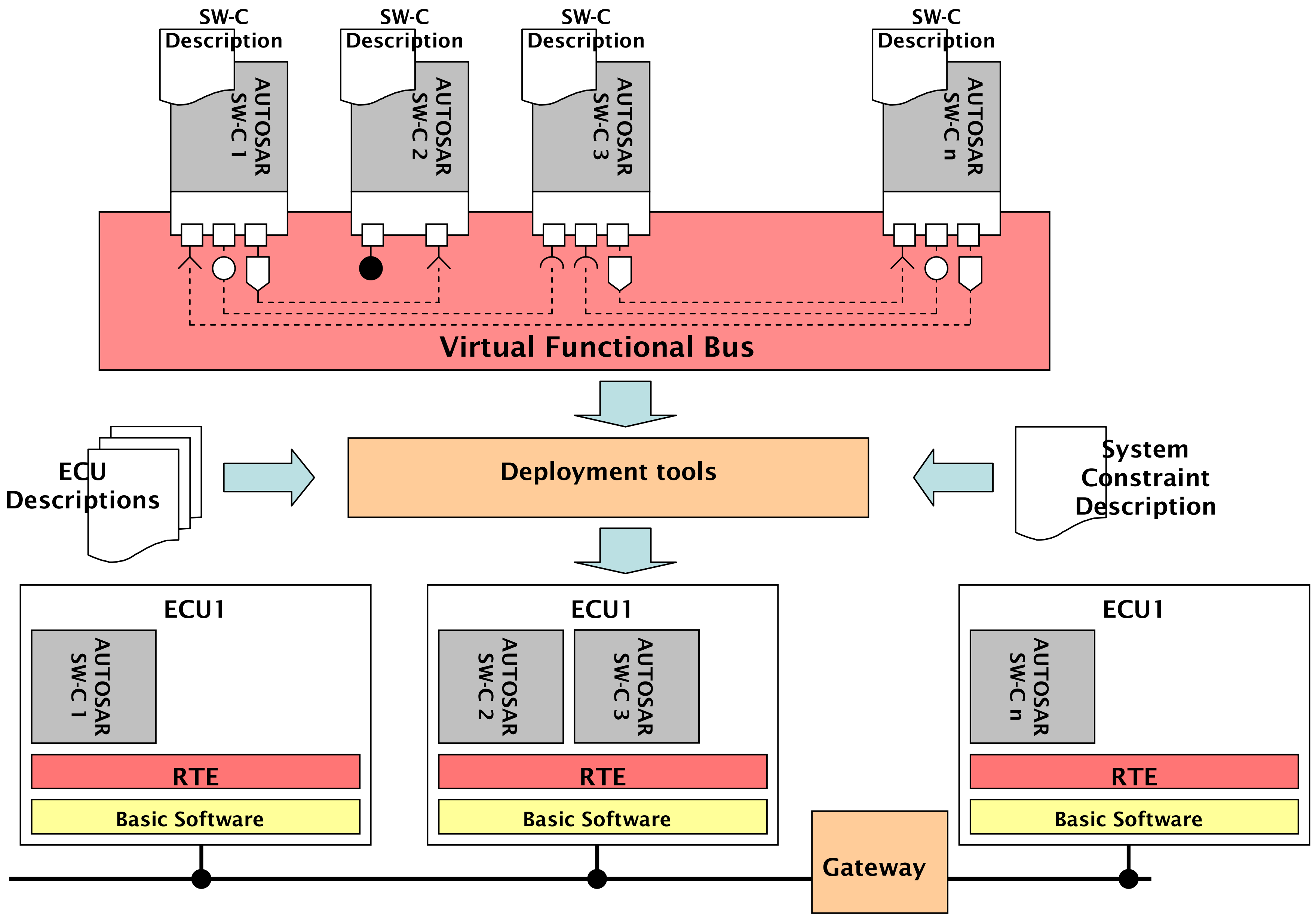
AUTOSAR – **AUT**omotive **O**pen **S**ystems **AR**chitecture

Middleware and system-level standard, jointly developed by automobile manufacturers, electronics and software suppliers and tool vendors.
More than 100 members

Motto: "*cooperate on standards, compete on implementations*"
Reality: current struggle between OEM and Tier1 suppliers

Target: facilitate portability, composability, integration of SW components over the lifetime of the vehicle

AUTOSAR Architecture



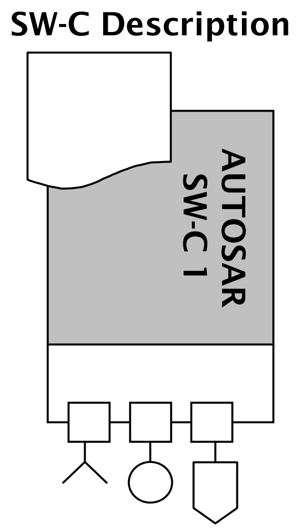
AUTOSAR Architecture

- **AUTOSAR SW-C**

  The AUTOSAR Software Components encapsulate an application which runs on the AUTOSAR infrastructure. The AUTOSAR SW-C have well-defined interfaces, which are described and standardized.

  - **SW-C Description**

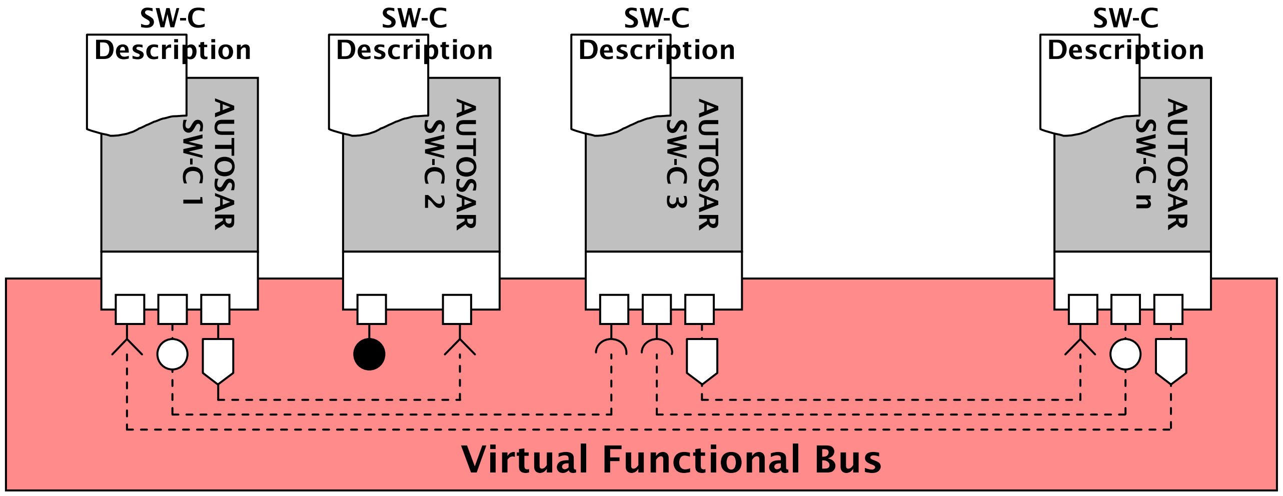
    For the interfaces as well as other aspects needed for the integration of the AUTOSAR Software Components, AUTOSAR provides a standard description format (SW-C Description).



AUTOSAR Architecture

- **Virtual Functional Bus (VFB)**

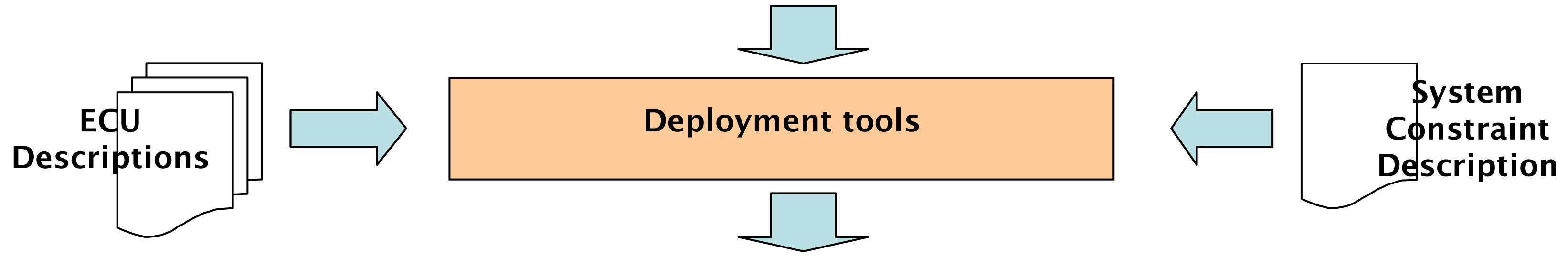
  The VFB is the sum of all communication mechanisms (and interfaces to the basic software) provided by AUTOSAR on an abstract (technology independent) level. When the connections for a concrete system are defined, the VFB allows a virtual integration in an early development phase.



AUTOSAR Architecture

- **System Constraint and ECU Descriptions**

  In order to integrate AUTOSAR SW-Components into a network of ECUs, AUTOSAR provides description formats for the system as well as for the resources and the configuration of the ECUs.

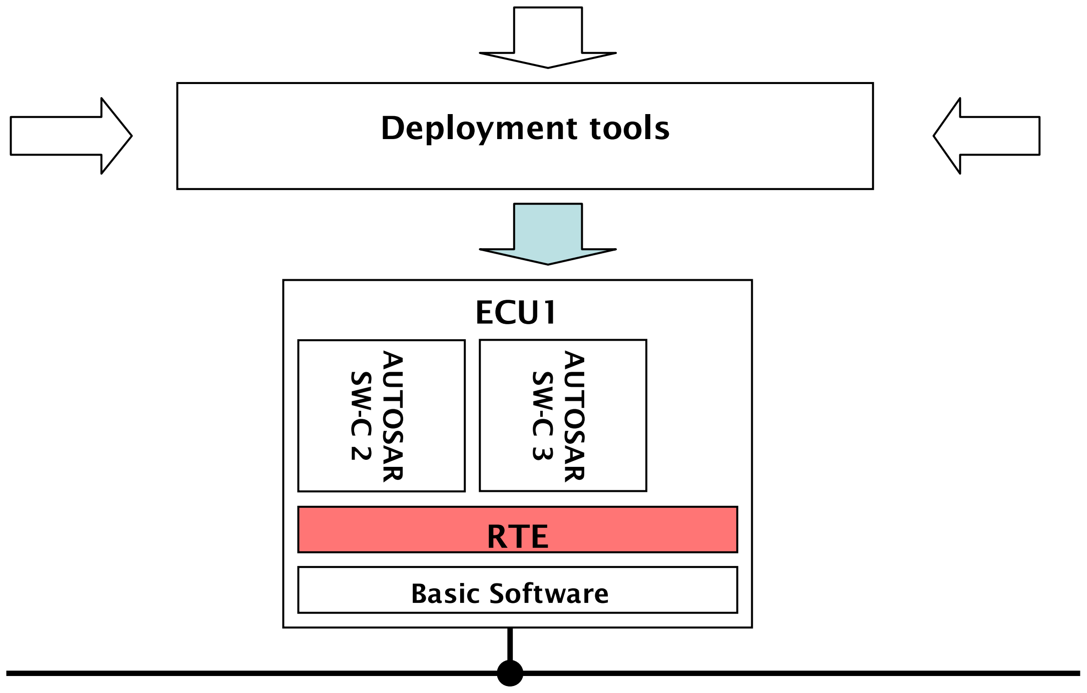


AUTOSAR Architecture: Mapping on ECUs

AUTOSAR defines the methodology and tool support to build a concrete system of ECUs. This includes the configuration and generation of the Runtime Environment (RTE) and the Basic Software (RTOS) on each ECU.

- **Runtime Environment (RTE)**

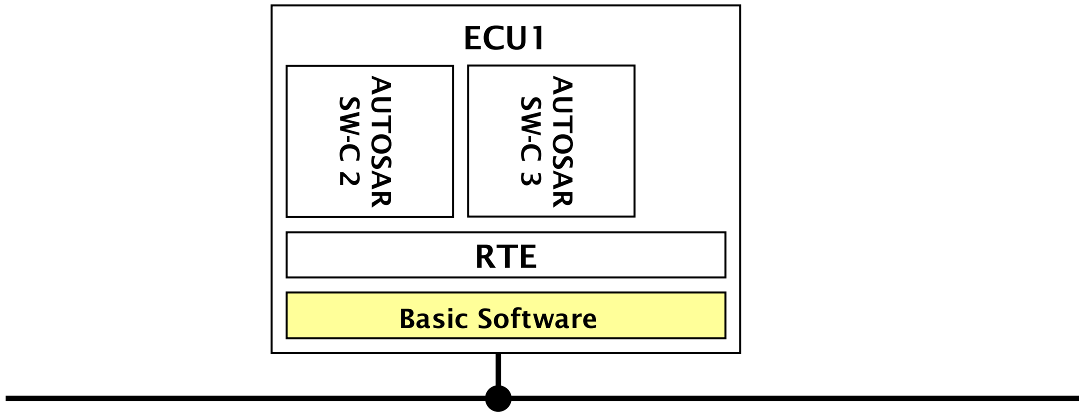
  From the viewpoint of the AUTOSAR Software Component, the RTE implements the VFB functionality on a specific ECU.



AUTOSAR Architecture

- **Basic Software**

  The Basic Software provides the infrastructure for execution on an ECU.



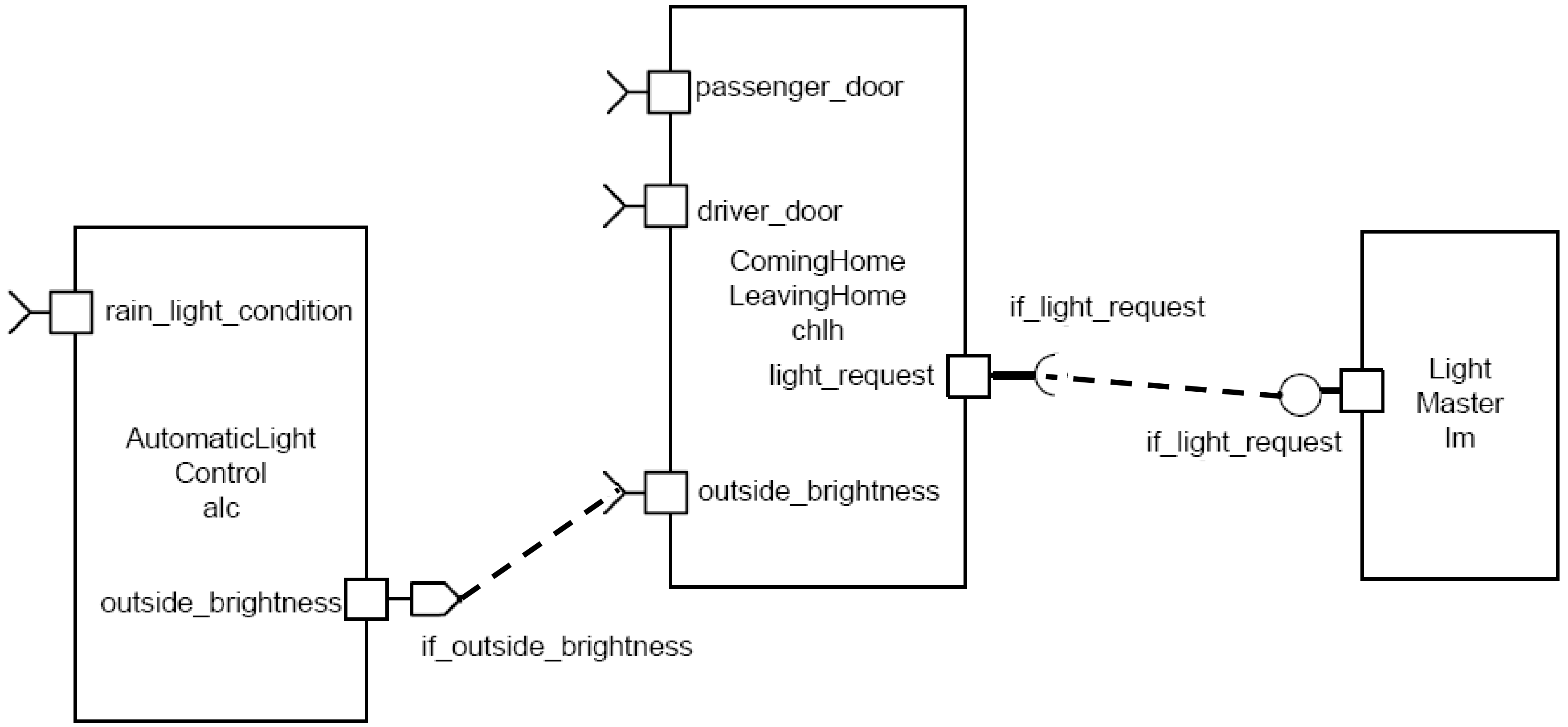
AUTOSAR Architecture

A fundamental concept of AUTOSAR is the separation between:

•**application** and

•**infrastructure**.

An application in AUTOSAR consists of Software Components interconnected by connectors



AUTOSAR Component

Each AUTOSAR Software Component encapsulates part of the functionality of the application.

- AUTOSAR does not prescribe the granularity of Software Components. Depending on the requirements of the application domain an AUTOSAR Software Component might be a small, reusable piece of functionality (such as a filter) or a larger block encapsulating an entire sybsystem.

The AUTOSAR Software Component is an "Atomic Software Component"

Atomic means that the each instance of an AUTOSAR Software Component is statically assigned to one ECU.

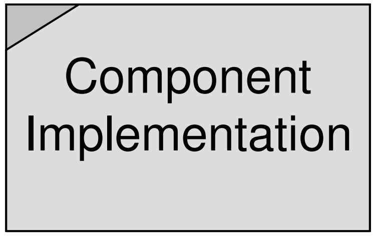
AUTOSAR Components

Shipping an AUTOSAR Software Component

A shipment of an AUTOSAR Software Component consists of

- a complete and formal ***Software Component Description*** which specifies how the infrastructure must be configured for the component, and
- an ***implementation*** of the component, which could be provided as "object code" or "source code".

Software Component Description



AUTOSAR Components: Description

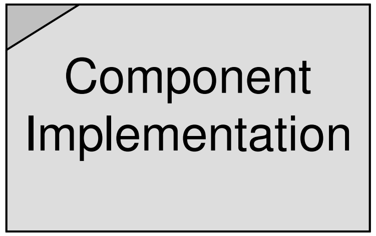
The AUTOSAR Software Component Description contains:

- the operations and data elements that the software component provides and requires
  - described using the PortInterface concept
- the requirements on the infrastructure,
- the resources needed by the software component (memory, CPU-time, etc.),
- information regarding the specific implementation of the software component.
- The structure and format of this description is called “software component template”.

AUTOSAR Components

A source code component implementation is independent from

- the type of microcontroller of the ECU and the type of ECU on which the component is mapped
  - The AUTOSAR infrastructure takes care of providing the software component with a standardized view on the ECU hardware
- the location of the other components with which it interacts. The component description defines the data or services that it provides or requires. The component doesn't know if they are provided from components on the same ECU or from components on a different ECU.
- the number of times a software component is instantiated in a system or within one ECU

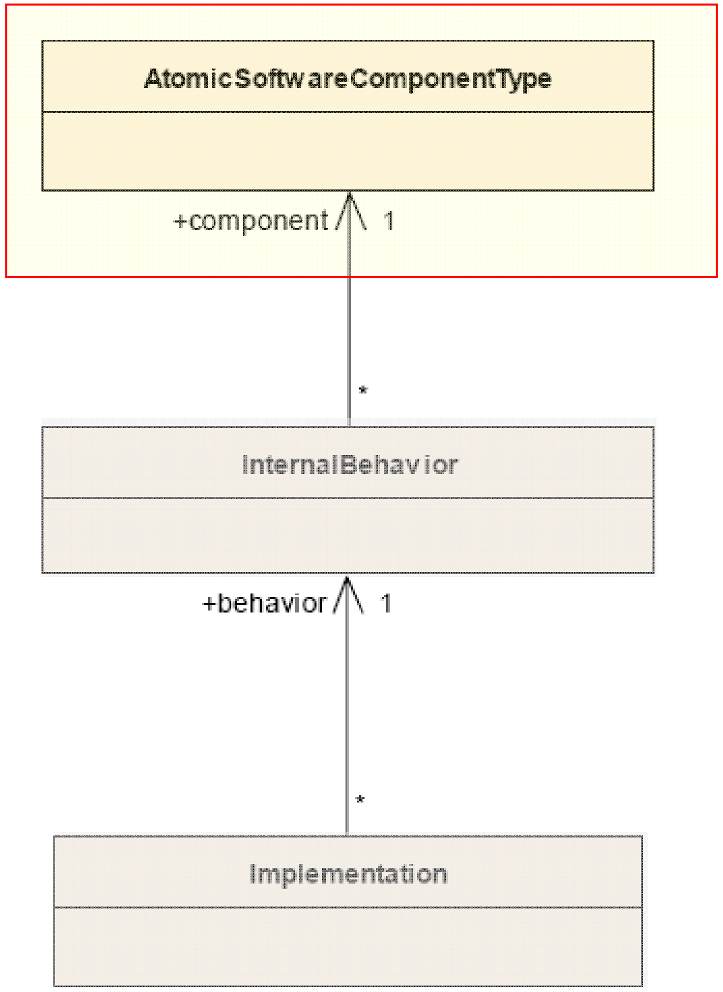


AUTOSAR Components: description levels

The highest (most abstract) description level is the Virtual Functional Bus.

Here components are described with the means of datatypes and interfaces, ports and connections between them, as well as hierarchical components. At this level, the fundamental communication properties of components and their communication relationships among each other are expressed.

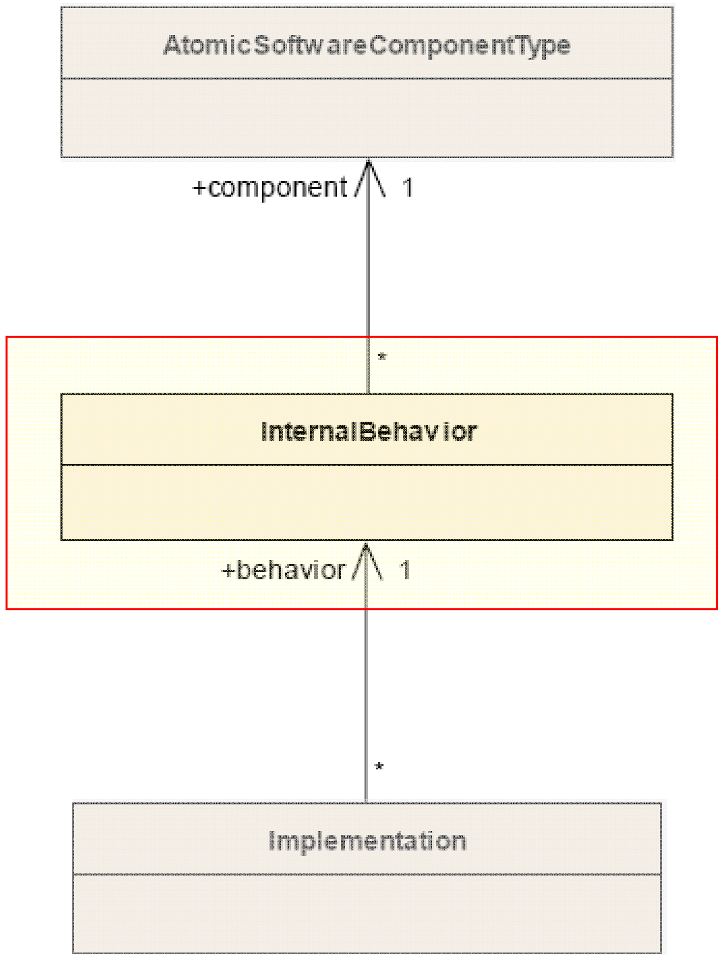
- **Software components**
- **Compositions**
- **Interfaces**



AUTOSAR Components

Description of components on RTE level: The middle level allows for behavior description of a given component. This behavior is expressed through RTE concepts, e.g. RTE events and in terms of schedulable units. For instance, for an operation defined in an interface on the VFB, the behavior specifies which of those units is activated as a consequence of the invocation of that operation.

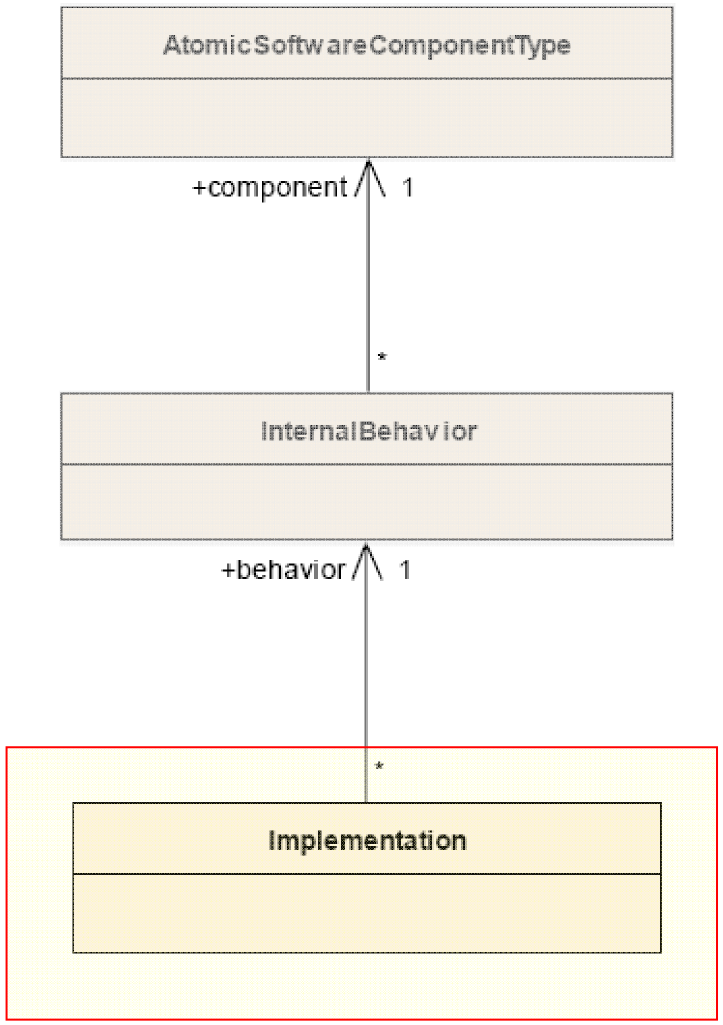
- **Runnables**
- **Events**
- **Interaction with the Run Time Environment**



AUTOSAR Components

Descriptions of components on implementation level: The lowest (most concrete) level of description specifies the implementation of a given behavior. More precisely, the schedulable units of such a behavior are mapped to code. The two layers above constrain the RTE API that a component is offered, the implementation now utilizes this API.

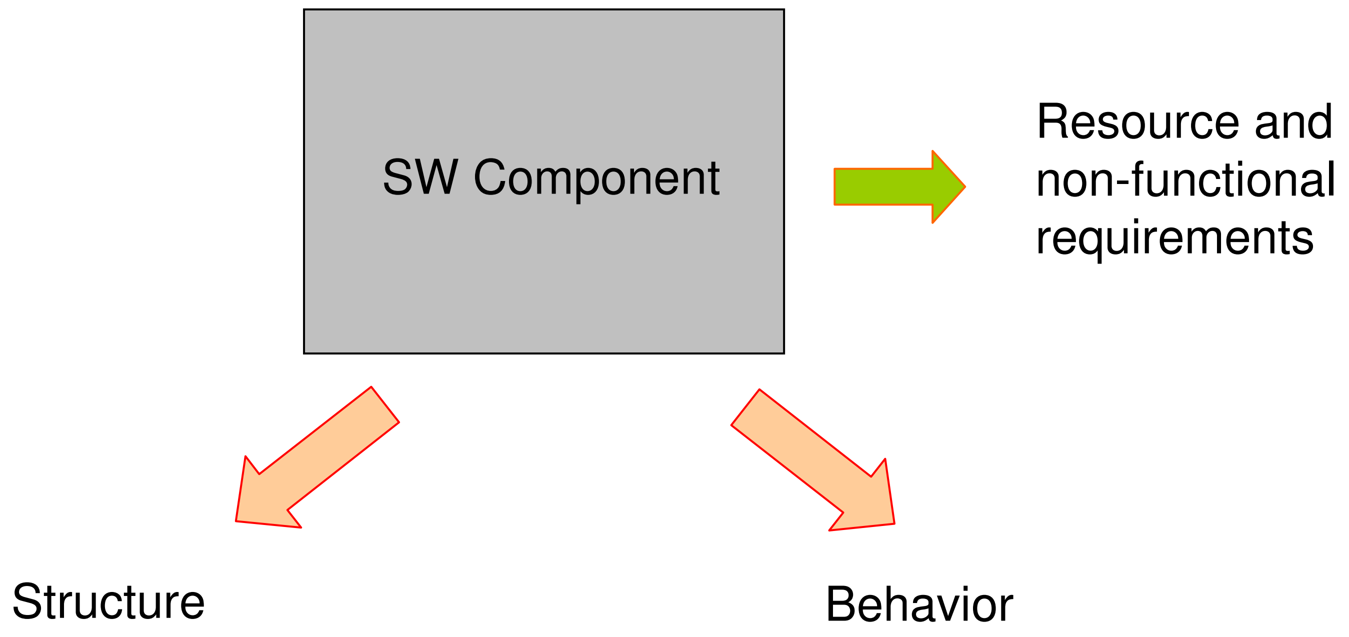
- **Component implementation**
- **Resource consumption of SW-Components**



Component-oriented design

What is a SW component?

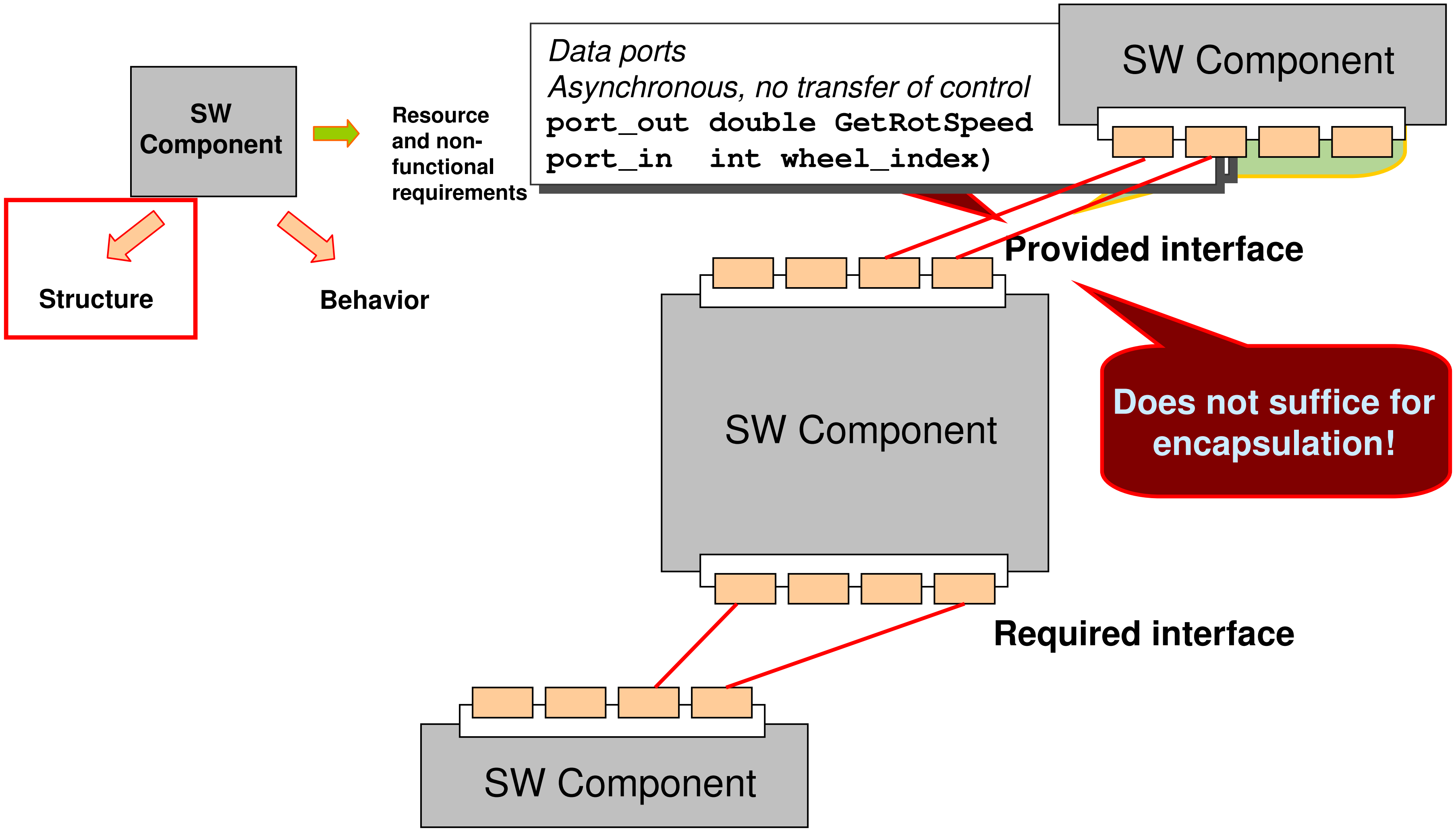
A reusable self-contained artefact implementing a function with given properties



Component-oriented design

Component structure

- Key concepts: information hiding and encapsulation



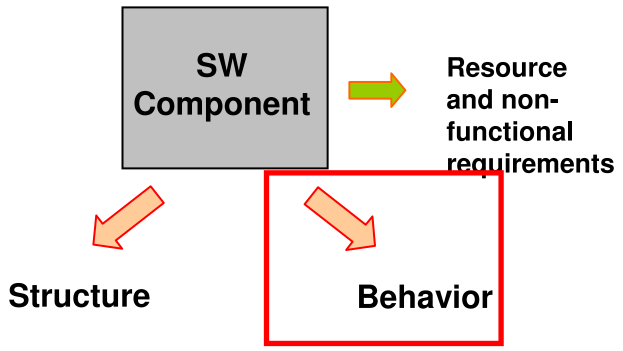
Data ports
Asynchronous, no transfer of control

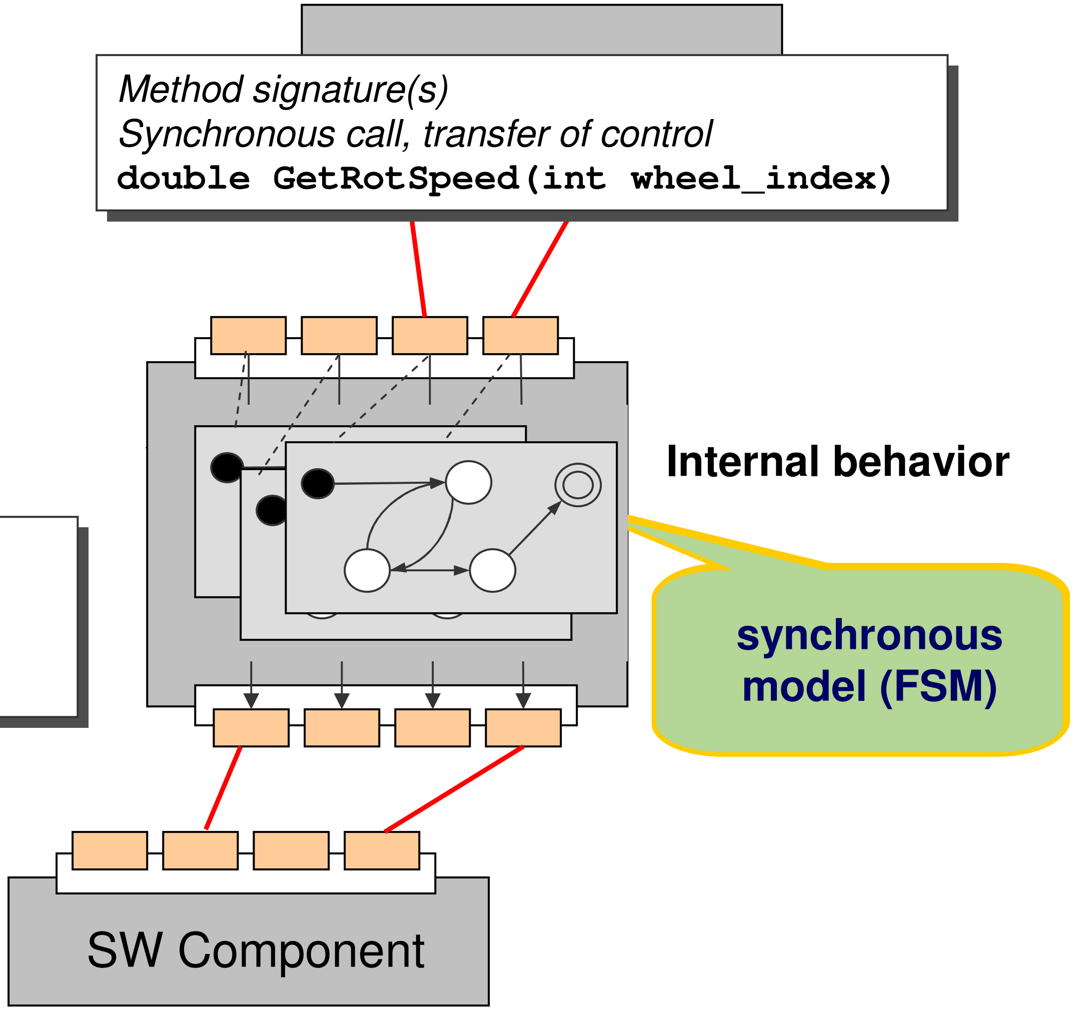
```
port_out double GetRotSpeed
port_in  int wheel_index)
```

Component-oriented design

Component structure

– Key concepts: information hiding and encapsulation





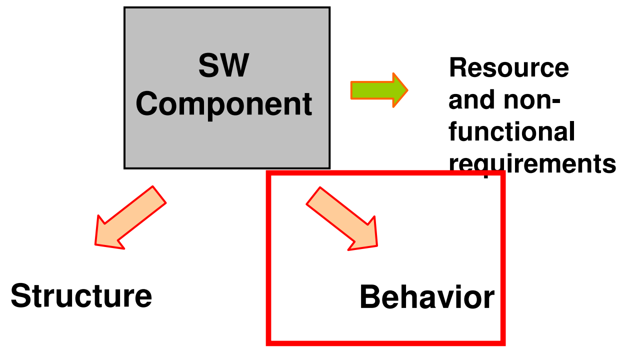
Method signature(s)
Synchronous call, transfer of control
`double GetRotSpeed(int wheel_index)`

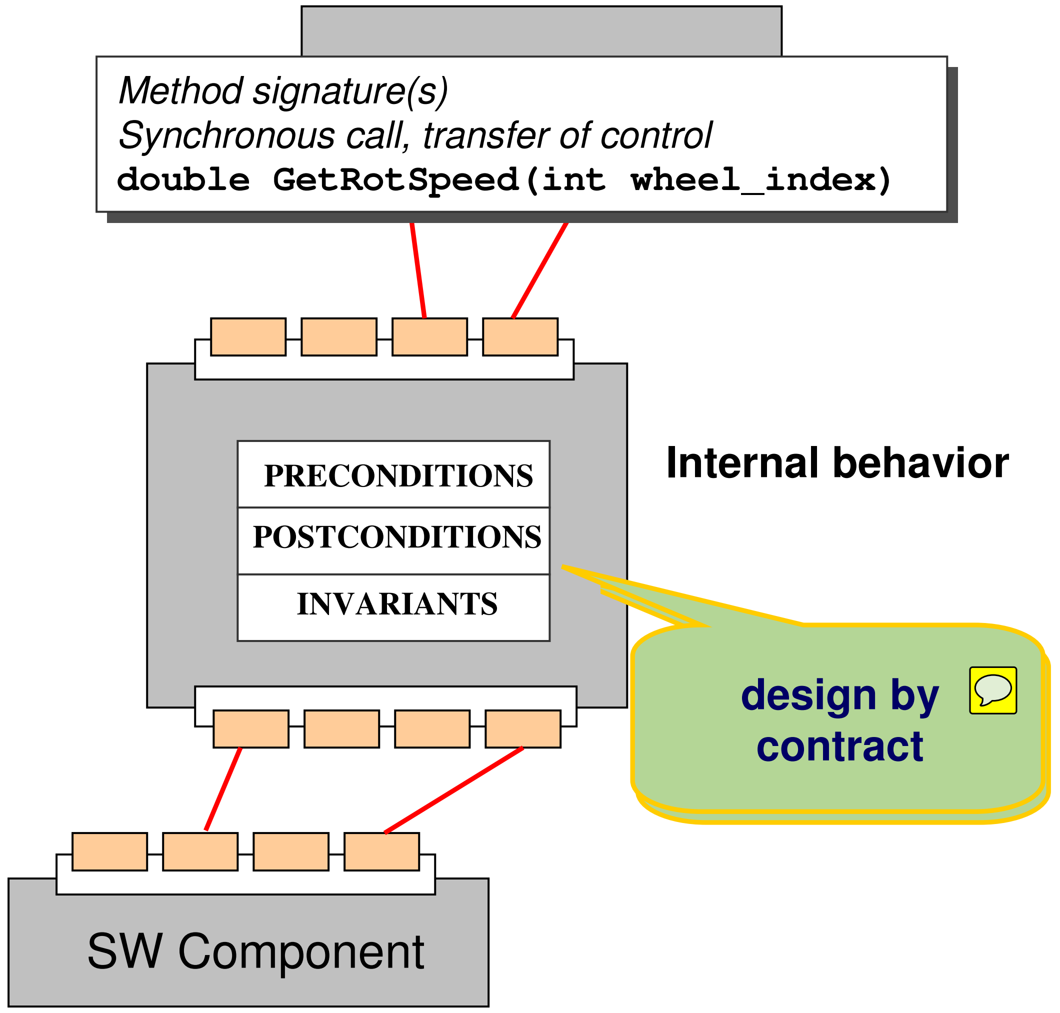
Data ports
Asynchronous, no transfer of control
```
port_out double GetRotSpeed
port_in  int wheel_index)
```

Component-oriented design

Component structure

– Key concepts: information hiding and encapsulation

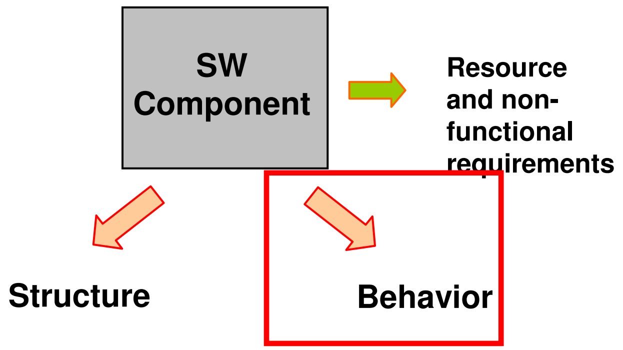


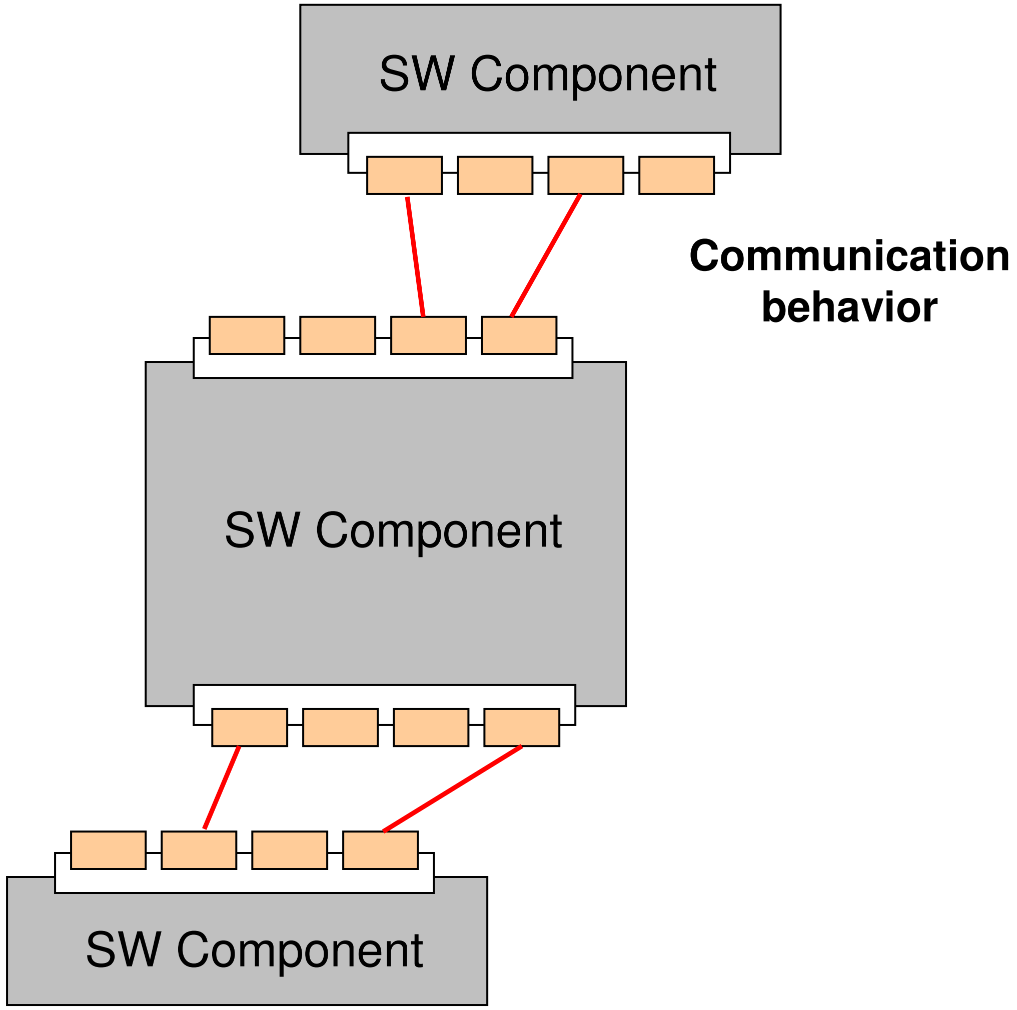


Component-oriented design

Component structure

– Key concepts: information hiding and encapsulation

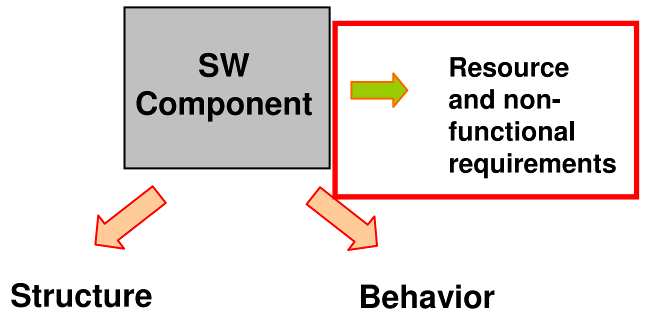


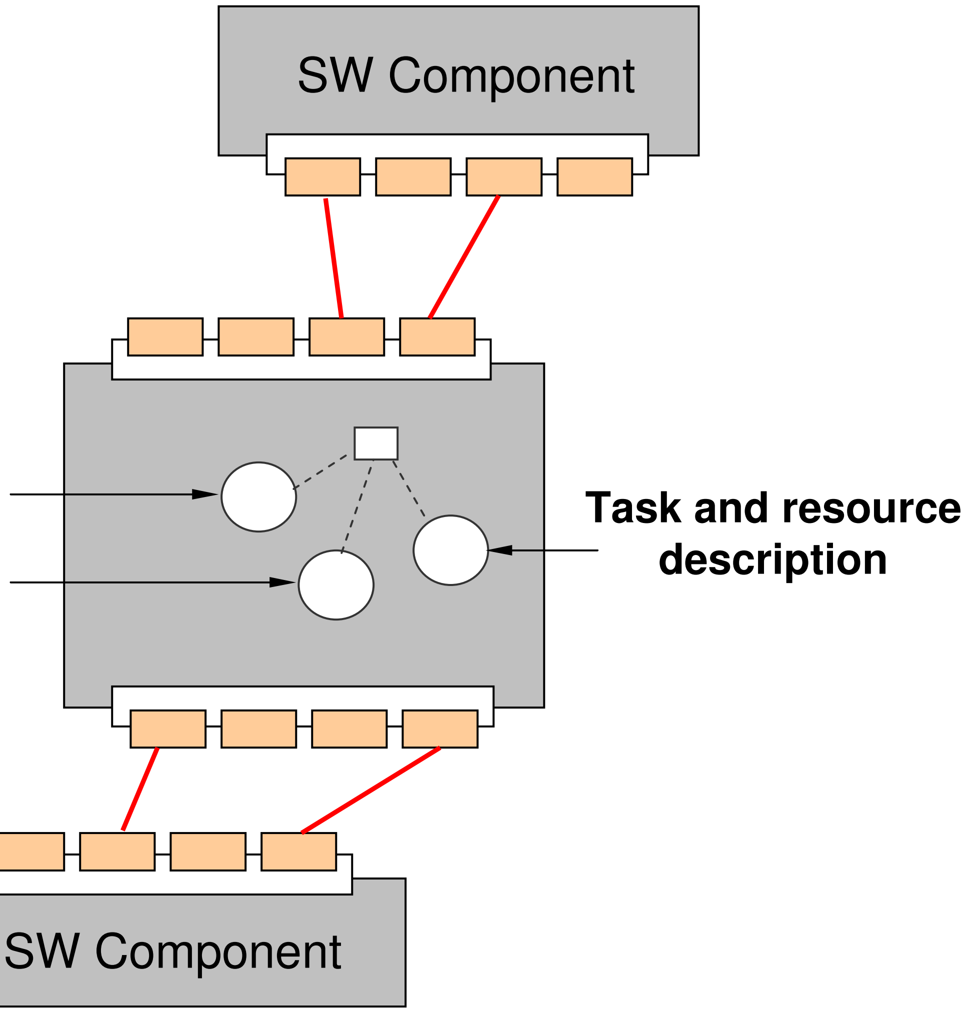


Component-oriented design

Component structure

– Key concepts: information hiding and encapsulation





The Virtual Function Bus (VFB)

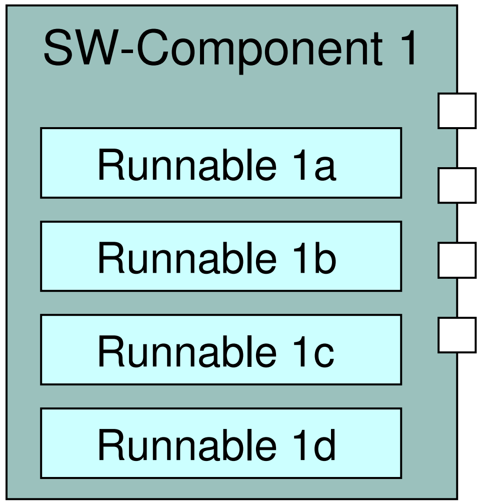
The virtual functional bus is the abstraction of the AUTOSAR Software Components interconnections of the entire vehicle. The communication between different software components and between software components and its environment (e.g. hardware driver, OS, services, etc.) can be specified independently of any underlying hardware (e.g. communication system). The functionality of the VFB is provided by communication patterns.

AUTOSAR Components: Runnable entities

Runnable Entities (also Runnable) are defined in the VFB specs.

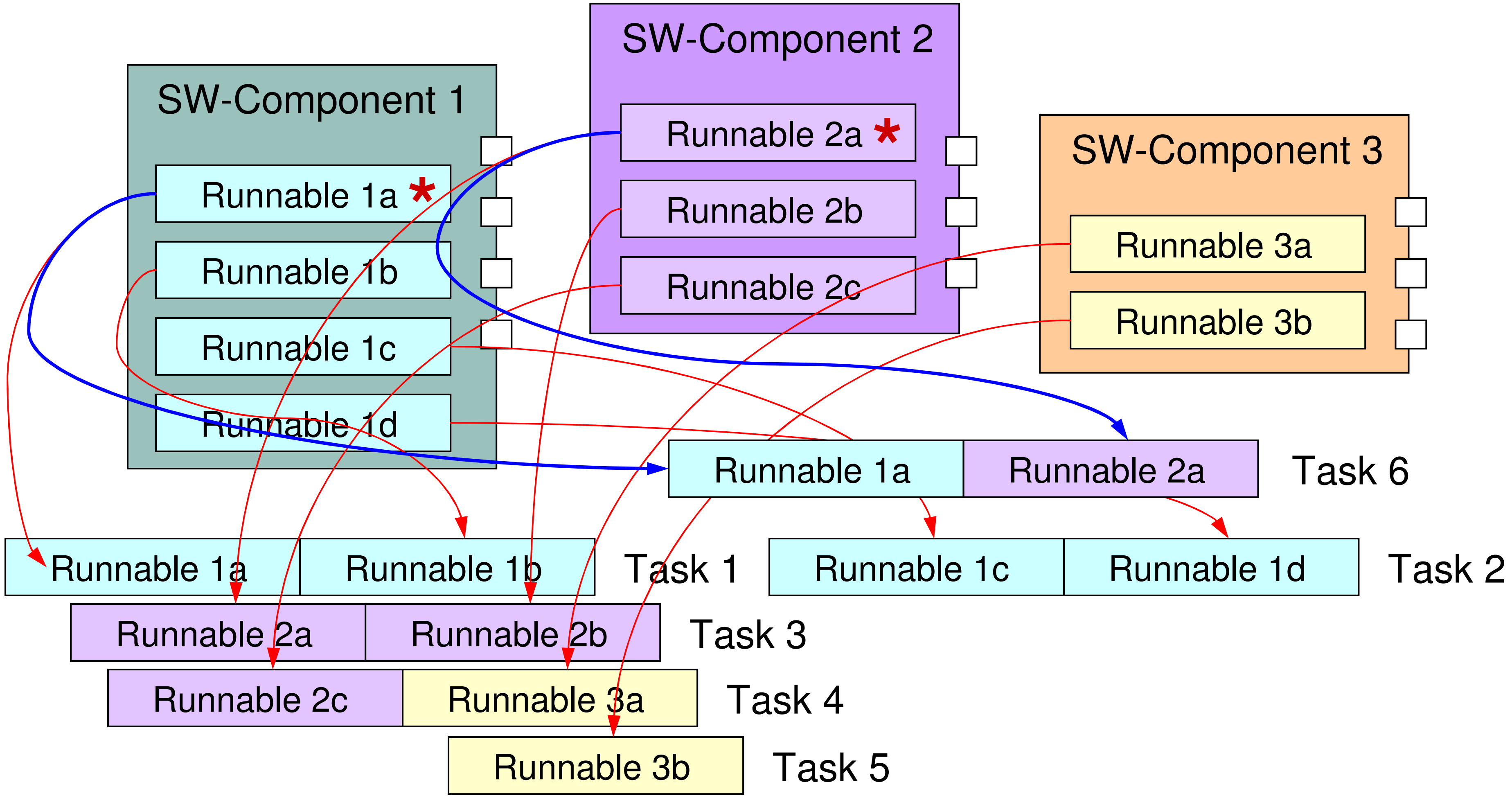
Runnable entities are the smallest code-fragments that are provided by the component and are (at least indirectly) a subject for scheduling by the operating system. An implementation of an atomic software-component has to provide an entry-point to code for each Runnable in its "InternalBehavior“.

- It is not possible for "CompositionType" to be referenced by "InternalBehavior". Only atomic software-components may have Runnables.

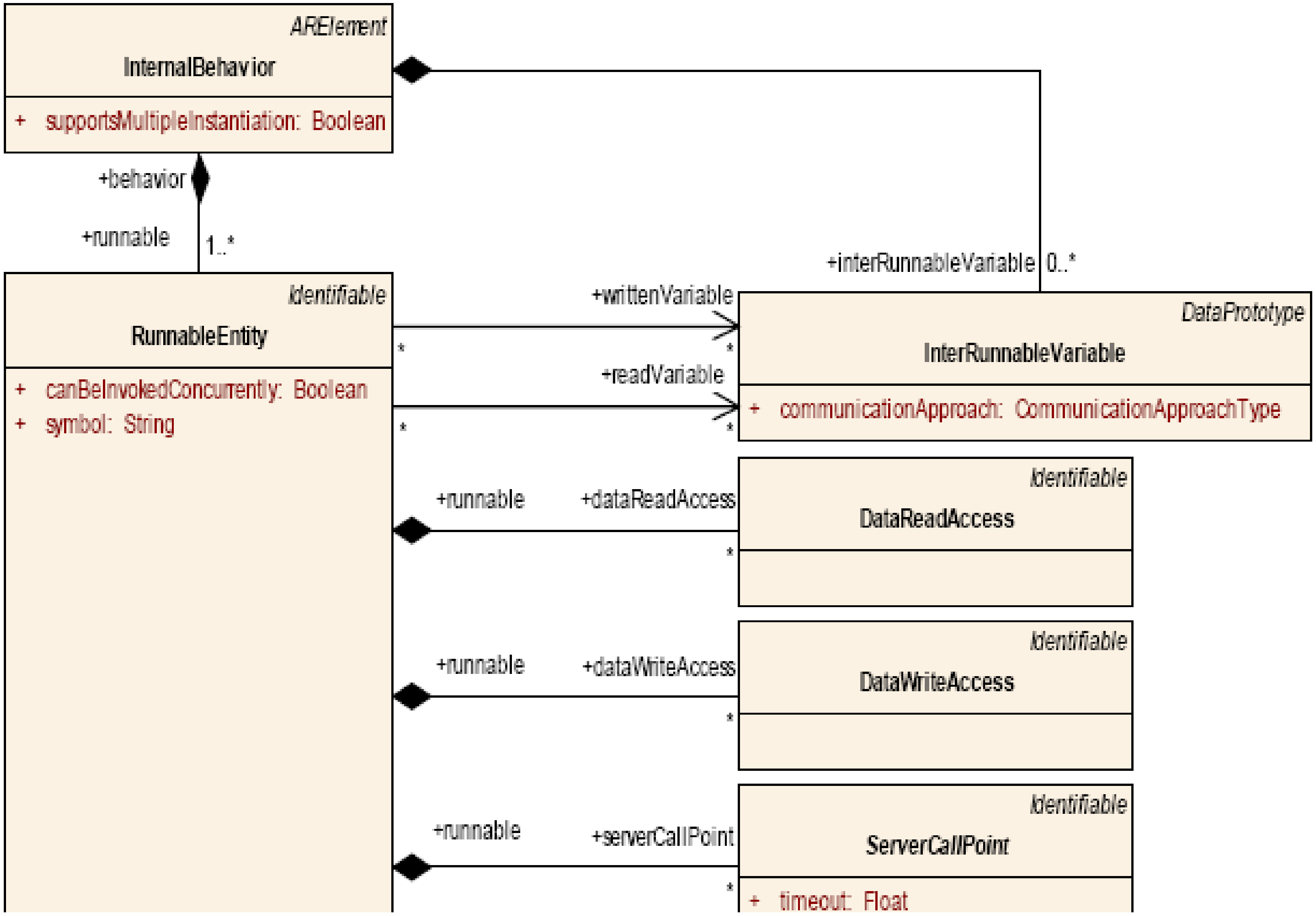


AUTOSAR Components: Runnable entities

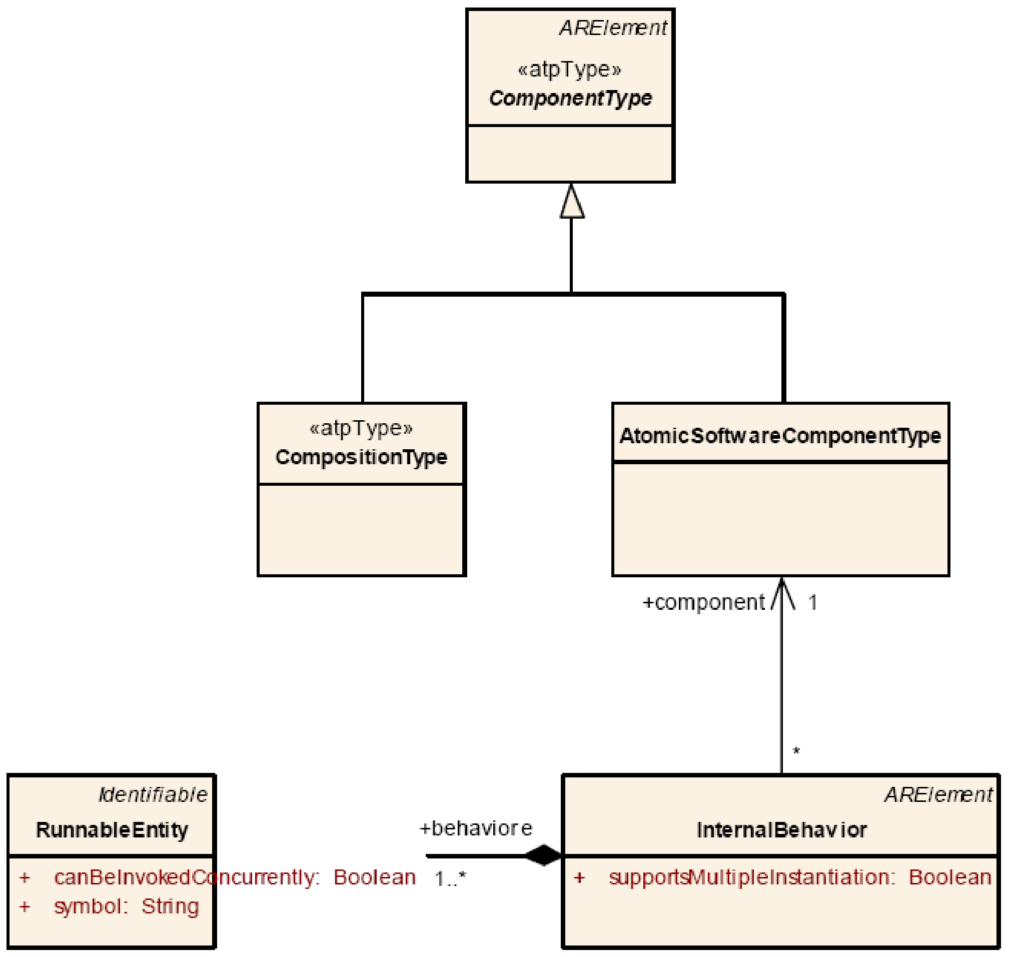
In most cases Runnables will not be scheduled individually but as parts of OS tasks.



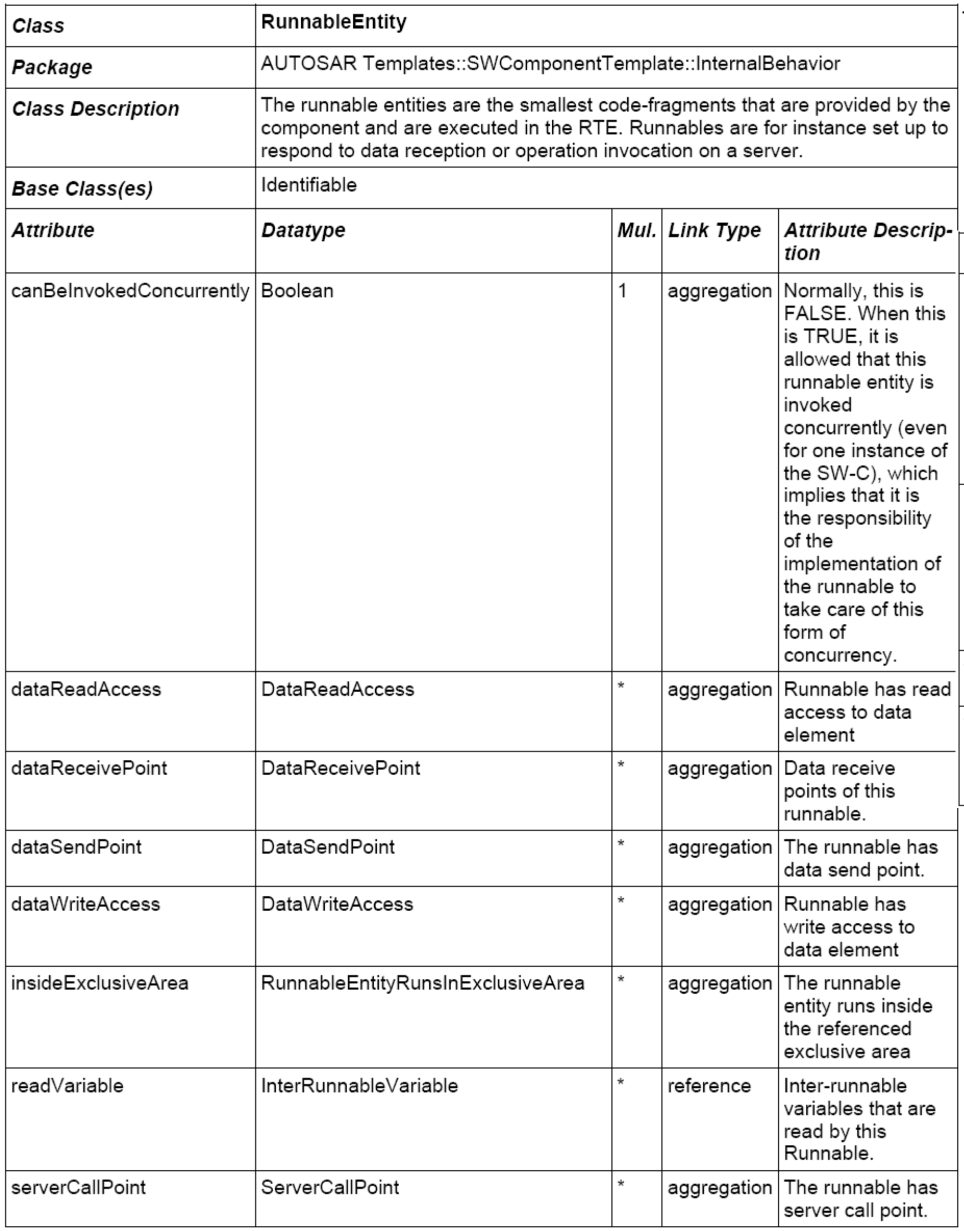
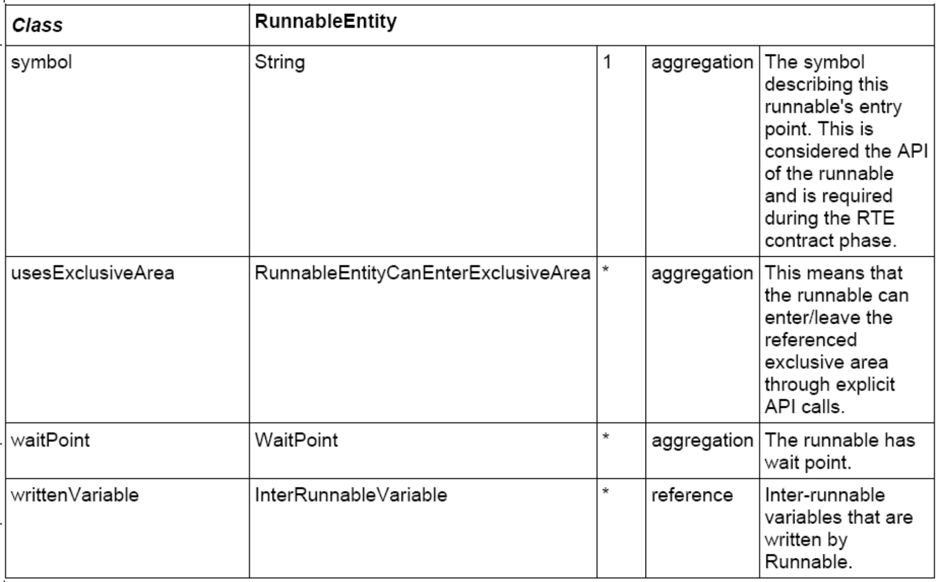
AUTOSAR Components: Runnable entities



AUTOSAR Components: Runnable entities



AUTOSAR Components: Runnable entities

Class	**RunnableEntity**			
Package	AUTOSAR Templates::SWComponentTemplate::InternalBehavior			
Class Description	The runnable entities are the smallest code-fragments that are provided by the component and are executed in the RTE. Runnables are for instance set up to respond to data reception or operation invocation on a server.			
Base Class(es)	Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
canBeInvokedConcurrently	Boolean	1	aggregation	Normally, this is FALSE. When this is TRUE, it is allowed that this runnable entity is invoked concurrently (even for one instance of the SW-C), which implies that it is the responsibility of the implementation of the runnable to take care of this form of concurrency.
dataReadAccess	DataReadAccess	*	aggregation	Runnable has read access to data element
dataReceivePoint	DataReceivePoint	*	aggregation	Data receive points of this runnable.
dataSendPoint	DataSendPoint	*	aggregation	The runnable has data send point.
dataWriteAccess	DataWriteAccess	*	aggregation	Runnable has write access to data element
insideExclusiveArea	RunnableEntityRunsInExclusiveArea	*	aggregation	The runnable entity runs inside the referenced exclusive area
readVariable	InterRunnableVariable	*	reference	Inter-runnable variables that are read by this Runnable.
serverCallPoint	ServerCallPoint	*	aggregation	The runnable has server call point.
Class	**RunnableEntity**			
symbol	String	1	aggregation	The symbol describing this runnable's entry point. This is considered the API of the runnable and is required during the RTE contract phase.
usesExclusiveArea	RunnableEntityCanEnterExclusiveArea	*	aggregation	This means that the runnable can enter/leave the referenced exclusive area through explicit API calls.
waitPoint	WaitPoint	*	aggregation	The runnable has wait point.
writtenVariable	InterRunnableVariable	*	reference	Inter-runnable variables that are written by Runnable.