Contents

1. Introduction

1.1 Intention of project

The scope of this project is aimed to help understand and identify the advantages and/or disadvantages of the diverse modeling languages.
Using different graphical models provided by a certain modeling language, one is able to make a precise specification of a complex system. Based on this specification and the fact that the process of modeling is abstract from the process of implementing, is the probability for misunderstandings, due to insufficient communication, very low.
In order to ensure meaningful assessment of the diverse modeling languages, a standardized software architecture need to be modeled, like the one offered by Automotive Open System Architecture (AUTOSAR). Because of the broad scope and variety of software layers presented in AUTOSAR architecture as show in Figure 1.1, only the memory stack is going to be modeled. The main reason for this decision is that the memory stack is to be found in every automotive unit[1].

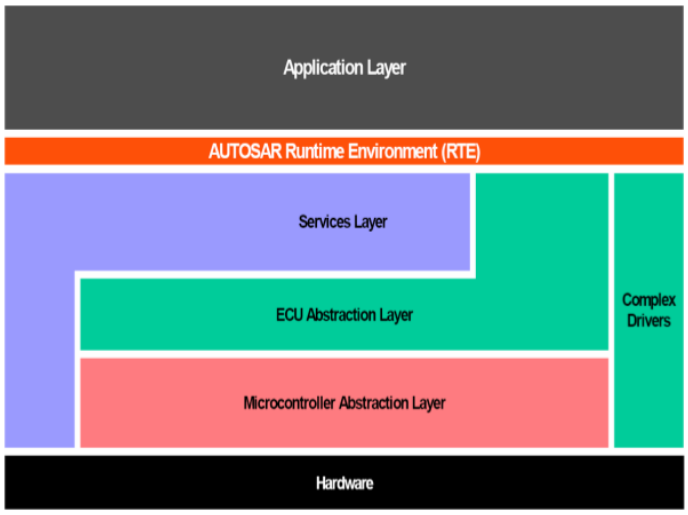


Figure 1.1: Layered Software Architecture - AUTOSAR

[1]Leitner Fl. Evaluation of the Matlab Simulink Design Verier versus the model checker, S. 24

1.2 Goals and success criteria of project

The project activities are going to be comparable to the base practices(BP) defined by the Software Process Improvement and Capability Determination(SPICE) model.

- BP1 - Software architectural design - 30.10.2009
- BP2 - Software requirements - 06.11.2009
- BP3 - Define interfaces - 20.11.2009
- BP4 - Dynamic behavior - 27.11.2009
- BP6 - Detailed design - 28.12.2009
- BP7 - Verification criteria - 11.01.2010
- BP8 - Verify Software design - 18.01.2010
- BP9 - Consistency and Bilateral Traceability - 28.01.2010

The BP5 and BP10 cannot be done in this setting due to missing work products. The project is successfully completed if the results are meaningful in industrial practice.

2. Case study

2.1 Overview

The design architecture used in AUTOSAR represents the layered pattern[1] using a component-based approach. One of the key benefits of this kind of architecture, which is applied extensively in software development, are the well-defined interfaces and strong dependency management.

As mention in 1.1 this case study will impact the modeling of the memory stack as shown in Figure 2.1.

[1] Automobil-Elektornik - January 2007 - S.28

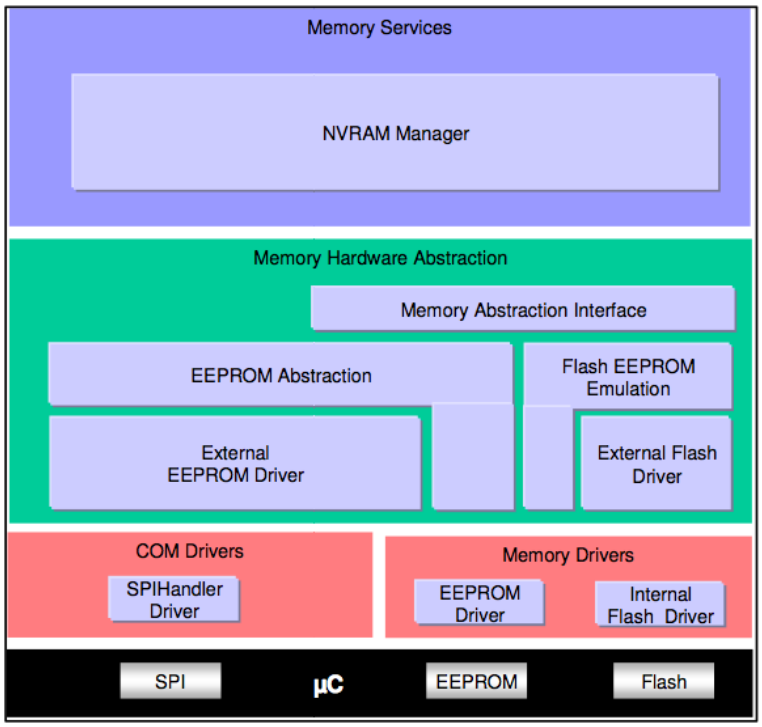


Figure 2.1: Memory stack

2.2 Requirements

2.2.1 Overview

Each module specific section contains a short functional description of the software module.Each module specific section contains a short functional description of the software module.

2.2.2 NVRAM-Manager

The NVRAM manager, as specified in [AUTOSAR_SRS_MemoryServeices.pdf], is highly scalable and hardware independent, thus making it usable in all domains of an automotive system. Providing service to manage the storage of data in all kinds of non-volatile memory and reliability mechanisms.

2.2.2.1 Acronyms and abbreviations

Basic Storage Object - The smallest entity of a NVRAM Block.

NVRAM Block - The entire structure, that administrate and stores a block of NV data.

NV data - The stored data in the non-volatile memory.

Block Management Type - Type of the NVRAM block. It depends on the individual composition of a NVRAM block in chunks of different mandatory or optional Basic Storage Objects and the subsequent handling of this NVRAM block.

RAM Block - A Basic Storage Object, that represents a part of a NVRAM block in RAM.

ROM Block - A Basic Storage Object that represents an optional part of a NVRAM block in ROM.

NV Block - A Basic Storage Object that represents a mandatory part of a NVRAM block in NV memory.

Administrative Block - A Basic Storage Object that resides in RAM. It contains any RAM data, that are necessary to manage the NVRAM block.

2.2.2.2 Functional requirements

2.2.2.3 Configuration

[BSW041] - Each application shall be enabled to declare and allocate the memory requirements at configuration time.

[BSW08534] - Two classes of RAM data blocks shall provide the service for data exchange between the NVRAM-Manager and SW-Components.

[BSW08528] - The basic storage of data and ROM configurable ROM defaults shall be provide from the native NVRAM block.

[BSW08529] - Data shall be provide in case of inconsistencies by the redundant NVRAM block, which is transparent to application.

[BSW08531] - To allow runtime selection of different datasets in ROM\'NVRAM, a dataset NVRAM shall be configurable to include default values.

[BSW08543] - The priority of each NVRAM block shall be statically configurable in different levels.

[BSW08009] - For each NVRAM block there shall be a static configuration of a default write protection, which shall be done by the NVRAM-Manager.

[BSW08549] - After a software update,the NVRAMmanager shall provide functionality to automatically initialize RAM data blocks with ROM defaults.

[BSW125] - For each block a notification shall be configurable, which will initiated after a completion of a read/write job.

[BSW135] - There shall be a configuration ID, that could be either statically assigned to the configuration or it can be calculated. This ID would be stored separately and should be used to determine the validity of the NVRAM contents.

[BSW08000] - The NVRAM manager shall be able to access multiple non-volatile memory devices, which can be of different type.

[BSW08001] - The NVRAM manager shall be check each block for consistency.

[BSW08538] - One kann statically configurate which blocks are going to be automatically loaded at startup of the NVRAM manager.

[BSW08546] - For each NVRAM block there shall be a configurable option to enforce meachnisms increasing integrity of RAM data.

2.2.2.4 Initialization

[BSW08533] - The NVRAM manager provides a service to check and load those NVRAM blocks, that are configured to have a permanent RAM data block.

2.2.2.5 Normal Operation

[BSW176] - The NVRAM manager could only access the non-volatile memory.

[BSW027] - The NVRAM manager provides an implicit way of accessing blocks in the NVRAM and in the RAM.

[BSW08014] - The RAM block would be defined as not continuos in the global RAM area.

[BSW013] - The NVRAM manager would be able to handle concurrent accesses to NVRAM memory.

[BSW016] - The NVRAM manager would be able to read out data associated with an NVRAM block from the non-volatile memory.

[BSW017] - The NVRAM manager would be able to store data associated with a NVRAM block in the non-volatile memory.

[BSW08541] - The NVRAM manager would accept and process any write request.

[BSW018] - The NVRAM manager would be able to restore aNVRAM block's associated data from RAM defaults.

[BSW08547] - The NVRAM manager is able to distinguish between explicitly invalidated and inconsistent data.

[BSW08548] - The NVRAM manager is able to request the default data from the application, if there is no ROM block available at configuration time.

[BSW08550] - The NVRAM manager is able to mark RAM data blocks as modified or unmodified. This service is configurable.

[BSW08545] - The NVRAM manager is able to mark the parmanent RAM data block of a NVRAM block valid. This configurable service could also update integrity information, if configured.

[BSW08011] - The NVRAM manager is able to invalidate an block of data in the non-volatile memory. This service is also configurable.

[BSW08544] - The NVRAM manager is able to erase the NV block or blocks associated with a NVRAM block, if configured.

[BSW08007] - The NVRAM manager could associate a dataset number to the corresponding RAM block.

[BSW08542] - The NVRAM manager provides a scheduling service for jobs. The highest priority job is to be processed first and if there is a job with the same priority level this shall be ececuted in FIFO order.

[BSW020] - The NVRAM manager is able to read out the status of read and write oprations.

[BSW127] - The NVRAM manager is able to enable and disable a write protection for each NVRAM block individually.

[BSW030] - The NVRAM manager is able to check for consistency and integrity of data saved in NVRAM during operation even in case of asynchronous reset or power loss.

[BSW034] - The NVRAM manager write accesses is concurrently executed to normal operation of the ECU.

2.2.2.6 Shutdown Operation

[BSW08535] - The NVRAM manager triggers update of integrity information and saving of permanent RAM data blocks to NV memory.

[BSW08540] - The NVRAM manager is to abort a ECU shutdown if a an ECU condition is detected.

2.2.2.7 Fault Operation

[BSW038] - The NVRAM manager is not going to report healable or recoverable errors to other software components.

[BSW0129] - If data corruption is detected and valid data can be derived, this is going to be done by the NVRAM manager.

[BSW08010] - If it is not possible for the NVRAM manager read data from NV into RAM, then it will copy the ROM default data to the data area of the corresponding RAM block.

[BSW08015] - After ECU reprogramming some of the NV blocks in the NVRAM are not going to be erased nor be replaced with the default ROM data after the first initialization.

2.2.2.8 Non-Functional Requirements

2.2.2.9 Hardware independence

[BSW011] - The NVRAM manager is going to access the underlying memory hardware via interfaces. Thus abstracting the memory hardware.

2.2.2.10 Usability

[BSW0130] - The NVRAM manager is providing information in MAP file format how many resources of RAM,ROM and NVRAM are being used.

2.2.3 Internal Flash Driver

This driver, as specified in [AUTOSAR_SRS_Flash_Driver.pdf], provides the ability to initialize, read, write and erase the internal Flash memory. The Flash driver has a built-in possibility to load the flash access code to RAM and execute the write or erase operations from there, if required.
The flash driver is only to be used by the Flash EEPROM emulation module for writing data, if the ECU is in application mode.

2.2.3.1 Acronyms and abbreviations

Such acronyms and abbreviations, that have a local scope do not appear in this section of the document, but in the local glossary.

2.2.3.2 Functional requirements

2.2.3.3 Configuration

[BSW12132] - The Flash driver is to be statically configurable.

[BSW12133] - The Flash shall publish diverse properties like: size in bytes, base address, physical memory segmentation and etc.

2.2.3.4 Normal Operation

[BSW12134] - With its asynchronous read function can the flash drive a certain data block read, starting at the requested flash address from the internal flash memory with the passed length.

[BSW12135] - With its asynchronous write function can the flash drive a certain data block write, starting at the requested flash address from the internal flash memory with the passed length. If the requested flash address is unaligned to the physical memory segment, this request shall be rejected with an error code.

[BSW12136] - With its asynchronous erase function can the flash drive a certain segments erase, starting at the requested flash address from the internal flash memory with the passed length. If the requested flash address is unaligned to the physical memory segment, this request shall be rejected with an error code.

[BSW12137] - With the synchronous cancel function can the flash driver stop any currently processed job.

[BSW12138] - The flash driver is able to return the job processing status using. It process is a synchronous function.

[BSW12141] - With the processing of writing the flash drive verifies, if statically configured, the written data by reading back from flash and comparing with the source data. If there are differences, those are going to be notified as errors.

[BSW12143] - The flash drive is able to complete only one job (erase or write) at time. If there is a new job request during an other job, the request will be rejected and handled as error. This detection is statically to be configured.

[BSW12144] - All processing jobs are nested in one processing job. Thus can the flash driver set the status variable.

[BSW12158] - A verification is needed to check if the addressed memory area is being erased, before the flash driver could write data to falsh memory.If this verification is falls, the process would be aborted with an error notification. This feature is to be statically configured.

[BSW12159] - The write and erase functions are providing with a service to check if the address parameters are within the calid configured address borders. If those are beyond the allowed borders, the process will be rejected with a error code.

[BSW12160] - If statically configured, the flash drive is going to check after an erase job, if the addressed block has been erased completely.

[BSW12193] - If an erase or write job is initiated, the flash drive loads the code needed to accesses the flash hardware to RAM. This feature is to be statically configured.

[BSW12194] - To keep the runtime as short as possible, the flash drive executes the code that accesses the flash hardware from RAM. This is only applicable if the code has been loaded to RAM.

[BSW13300] - If statically configured, the flash drive will remove the accesses code from RAM after the erase or write job is been finished or canceled. This is only then necessary if the flash driver has loaded that code to RAM during start of an erase or write job.

[BSW13301] - With its asynchronous compare function can the flash drive a certain section in the memory with a one in flash memory compare, starting at the requested flash address from the internal flash memory with the passed length.

[BSW13302] - The flash driver is capable to synchronous switch the operation mode from normal and fast memory access, and visa versa.

[BSW13303] - Each read job is limit to the configured default block size.Thus in normal mode, one cycle a job can read only a limit size of data from the memory.

[BSW13304] - If in fast mode, than each read job is going to read the maximum block size.

2.2.3.5 Non-Functional Requirements

[BSW12083] - The HIS specification is used as basis for specifying the flash driver.

[BSW12145] - If a write operation is executed, then the flash driver processes in one step only as much data as the flash hardware can handle. If a read operation is executed, then the flash driver processes in one step as much as a user has defined.

2.2.4 External Flash Driver

The external flash drive, as specified in [AUTOSAR_SRS_Flash_Driver.pdf], has the same functional scope as an internal flash drive, providing service for initialization, reading, writing and erasing the internal flash memory.

2.2.4.1 Acronyms and abbreviations

Such acronyms and abbreviations, that have a local scope do not appear in this section of the document, but in the local glossary.

2.2.4.2 Functional requirements

2.2.4.3 General

[BSW12147] - Scope of requirements for the external flash drive are the same as like for an internal flash drive.

2.2.4.4 Configuration

[BSW12182] - The external flash drive can besides the basic configuration parameters as well as configuring the static parameters: expected hardware ID and maximal read access blocking time.

2.2.4.5 Fault Operation

[BSW12107] - The initialization function of the external flash driver checks if there is a mismatch between configured flash typed and hardware flash ID. If case of mismatch, this will be reported to the error manager.

2.2.4.6 Non-Functional Requirements

[BSW12148] - The APIs from the internal and external flash drivers are semantically indentical.

[BSW12149] - To insure the reuse of external flash drive across multiple microcontrollers the source code is independent from the underlying microcontroller.

[BSW12184] - The flash driver limits the read access blocking time to the configured time.

2.2.5 Internal EEPROM Driver

The internal EEPROM Driver, as specified in [AUTOSAR_SRS_EEPROM_Driver.pdf], has the functions for initialization, reading, writing and erasing to or from internal EEPROM.

2.2.5.1 Acronyms and abbreviations

Such acronyms and abbreviations, that have a local scope do not appear in this section of the document, but in the local glossary.

2.2.5.2 Functional requirements

2.2.5.3 Configuration

[BSW096] - The EEPROM driver has a statically configurable parameters like: base address, size, maximum block size maximum read block size and call cycle of cycle job processing function for read, write and erase.

[BSW12071] - The driver description contains the following parameters: total physical EEPROM size, value of erased EEPROM cell, the size of one EEPROM cell and physical memory segmentation.

2.2.5.4 Normal Operation

[BSW087] - With the asynchronous read function can the EEPROM driver read a certain data block starting from the requested address with the passed length from the internal EPPROM.

[BSW088] - With the asynchronous write function can the EEPROM driver write a certain data block starting from the requested address with the passed length from the internal EPPROM. In case that the addressed cell is not empty, an erase operation is executed automatically, before the write one. If there is a mismatch in the length between the erased block and the write requested block, then the driver will buffer and rewrite those data in the block additionally.

[BSW089] - With the asynchronous erase function can the EEPROM driver erase a certain data block starting from the requested address with the passed length from the internal EPPROM. If there is a mismatch in the length between the erased block and the erase requested block, then the driver will buffer and rewrite the data in the block additionally.

[BSW090] - The EEPROM driver provides a synchronous cancel function, that stops the currently processed job.

[BSW091] - With the synchronous status function can the EEPROM driver return the job processing status.

[BSW092] - The EEPROM driver compares the data to be written with data to be erased, if at least one data value of the affected erasable block is different, than the data will be written, if statically configured.

[BSW094] - The memory segmentation is handle by the EPPROM driver. If necessary the drive could resolve the physical EEPROM block size and segment border by executing read-modify-write operations.

[BSW095] - The EPPROM driver can handle only one job at time. If there are meanwhile other jobs requests, those are going to be rejected and handle as errors. This detection is to be statically configured.

[BSW12047] - All processing jobs are nested in one processing job. Thus can the EEPROM driver set the status variable.If supported by hardware, this function can be called from an interrupt, else this function should be handled by a dedicated module.

[BSW12072] - If the EPPROM drive is operating in fast mode, then one cycle of the job processing function is limited to the configured maximum block size.

[BSW12091] - With the asynchronous compare function can the EEPROM driver compare a certain section in memory with a section inEPPROM with the passed length.

[BSW12156] - With the synchronous switch function can the EEPROM driver change the operation mode from normal into fast and visa versa.

[BSW12157] - If the EEPROM driver is operating in normal mode, then one cycle of the job processing function is limited to the configured default block size.

2.2.5.5 Non-Functional Requirements

[BSW12050] - In one step can the EEPROM drive process only as much data as the EEPROM hardware can handle or as much as defined by the user.

2.2.6 External EEPROM Driver

The external EEPROM Driver, as specified in [AUTOSAR_SRS_EEPROM_Driver.pdf], has the functions for initialization, reading, writing and erasing to or from external EEPROM.

2.2.6.1 Acronyms and abbreviations

Such acronyms and abbreviations, that have a local scope do not appear in this section of the document, but in the local glossary.

2.2.6.2 Functional requirements

2.2.6.3 General

[BSW12051] - Scope of requirements for the external EEPROM drive are the same as like for an internal EEPROM drive.

2.2.6.4 Configuration

[BSW12164] - The SPI EEPROM driver allows the static configuration of the required SPI parameters specified by the SPI handler.

2.2.6.5 Normal Operation

[BSW12124] - Depending on the EEPROM mode, the external SPI EEPROM device shall access as: in normal mode with single byte or word mode, in fast mode with burst mode.

2.2.6.6 Non-Functional Requirements

[BSW12052] - The APIs from the internal and external EEPROM Drives are semantically indentical.

[BSW12053] - To insure the reuse of external EEPROM drive across multiple microcontrollers the source code is independent from the underlying microcontroller.

2.2.7 Memory Abstraction Modules

The EEPROM Abstraction Layer, as specified in [AUTOSAR_SRS_MemHw_AbstractionLayer.pdf], extends the EEPROM(EA) driver so that it provides upper layers with a virtual segmentation on a linear address space and a virtually limitless number of erase and write cycles. It also provides the same functionality as an EEPROM driver.
The flash EEPROM emulator(FEE) emulates the behavior of the EEPROM abstraction layer on flash memory technology, having the same functional scope and API as the EEPROM abstraction layer.

2.2.7.1 Acronyms and abbreviations

Such acronyms and abbreviations, that have a local scope do not appear in this section of the document, but in the local glossary.

2.2.7.2 Functional requirements

2.2.7.3 Configuration

[BSW14001] - The configuration of the start and end addresses of logical block is done by the FEE and EA modules.

[BSW14002] - The required number of write cycles for each logical block are going to be determine by the FEE and EA modules.

[BSW14003] - The maximal amount of time that the module is being blocked by internal management operations is to be configured by the EA and FEE modules. This operation takes longer as expected, then this is going to be split up in several parts.

[BSW14004] - The configuration done by the FEE and EA modules of logical blocks is only limited by the memory available on the respective device.

[BSW14026] - The two blocks numbers 0x0000 and 0xFFFF are not to be used by the memory abstraction module.

[BSW14027] - The internal management overhead of the FEE and EA modules is going to be publish per logical block.

[BSW14033] - The internal management overhead is going to be publish by the FEE and EA modules per page or zero if there is no overhead for managing pages.

2.2.7.4 Normal Operation

[BSW14005] - There are going to be upper layers with a virtual 32bit address space provided by the FEE and EA modules.

[BSW14006] - Every write block or erase block request starts at address offset zero.

[BSW14007] - The read operation can start at any given memory address.

[BSW14008] - There is not check done by the FEE or EA for the address offset for a read operation.

[BSW14009] - An unambiguous conversion between the logical linear addresses and the addresses used to access the underlying flash memory is provided by the FEE and EA modules.

[BSW14010] - The write operation provided by the FEE and EA module operates only on complete configured logical blocks.

[BSW14012] - The FEE or EA module provides sufficient mechanisms to spread the write requests over a bigger memory area if the configured number of cycle for a logical block exceeds the number provided by the underlying physical device.

[BSW14013] - Writing of immediate data must not be delayed by internal management operations nor by erasing the memory area to be written to. If there are operation blocking the writing, then these are going to be interrupted.

[BSW14028] - The invalidation of a certain logical block is provided by the FEE and EA modules.

[BSW14029] - A part or all of a logical block can be read.

[BSW14031] - Every asynchronous operation can be cancelled by the canceling service provided by the FEE and EA modules.

[BSW14032] - The erase operation from the FEE and EA modules is only available when that service operates on complete logical blocks containing immediate data.

2.2.7.5 Fault Operation

[BSW14014] - Data inconsistencies due to aborted or interrupted write operation is to be detected by the FEE and EA modules.
[BSW14015] - If there are data inconsistencies, those are going to be marked and reported just once to the DEM.

[BSW14016] - After an interrupted or cancelled write operation the data of that block is inconsistent. This inconsistency is detected on the next read access, thus it does not return any data.

2.2.7.6 Non-Functional Requirements

[BSW14017] - The functional scope of an EEPROM driver is extended by the EEPROM abstraction layer.

[BSW14018] - The functional scope of an internal flash driver is extended by the flash EEPROM emulation.

2.2.8 Memory Abstraction Interface

Provides upper layers with a virtual segmentation on a uniform linear address space.

2.2.8.1 Acronyms and abbreviations

2.2.8.2 Functional requirements

2.2.8.3 General

[BSW14019] - Provides those services that using uniform access to those APIs that are required for usage within the NVRAM manager.

[BSW14020] - Using device index can the memory abstraction interface select the FEE or EA modules.

2.2.8.4 Configuration

[BSW14021] - The configuration of the number of underlying memory abstraction modules is allow in the pre-comple time.

2.2.8.5 Normal Operation

[BSW14022] - The memory abstraction interface preserves only the functionality of the underlying memory abstraction module.

2.2.8.6 Fault Operation

[BSW14023] - The parameters that are going to be check by the memory abstraction interface are those that are used within the interface itself.

2.2.8.7 Non-Functional Requirements

2.2.8.8 Timing Requirements

[BSW14024] - By mapping of the memory abstraction interface API to the memory abstraction modules API can the timing behavior be preserved.

2.2.8.9 Resource Usage

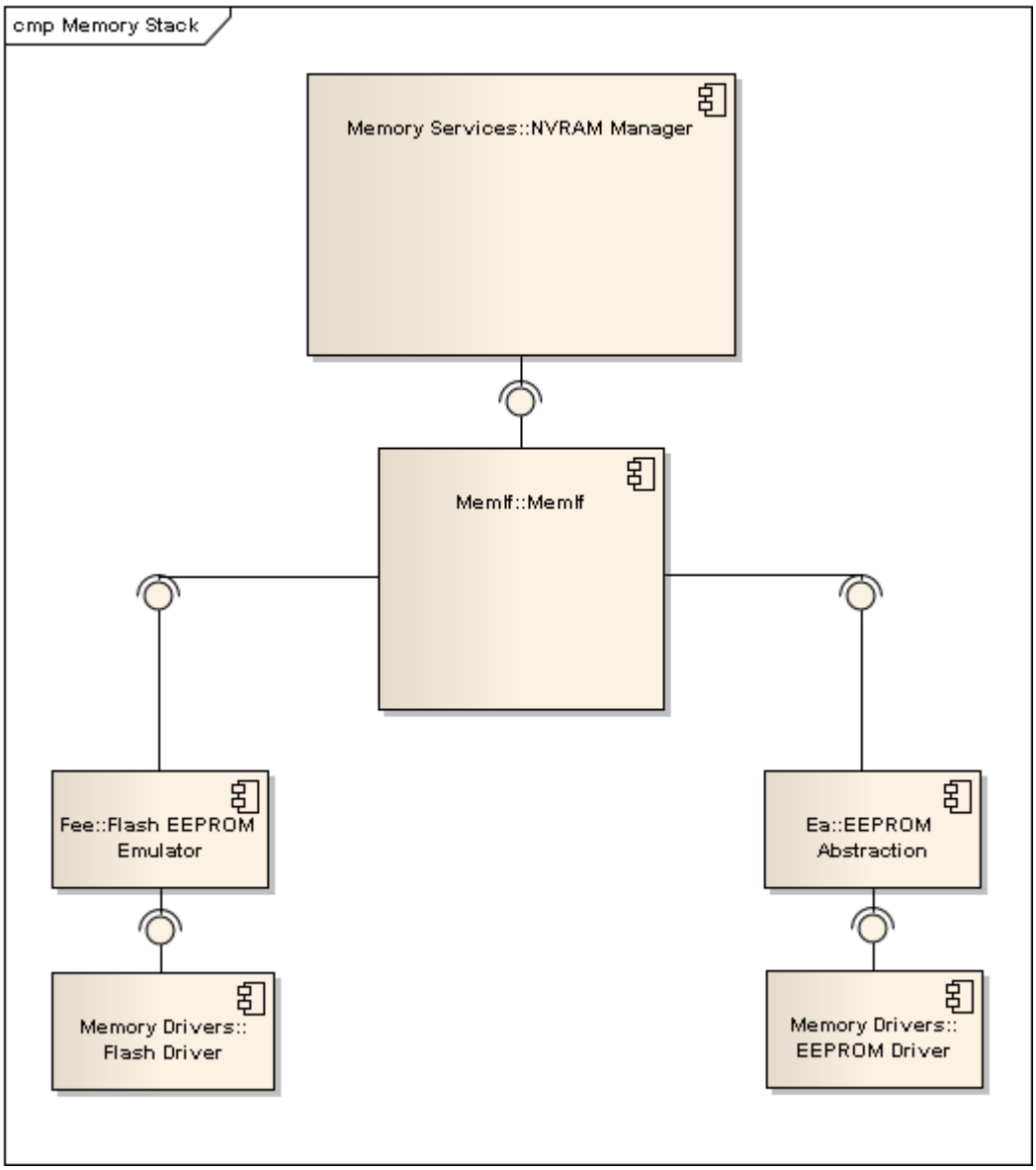
[BSW14025] - The memory abstraction interface is in such a way implemented that almost no memory and runtime overhead is caused by the implementation of this layer.

2.3 Modelling with UML

2.3.1 Software architecture

2.3.1.1 Component segmentation and description of interfaces

The purpose of the following diagram is to show the structural relationship between the components of the memory stack, to provide a high-level architectural view and to give a rough information about the components.



2.4 Modelling with SysML

2.4.1 Software architecture

2.4.1.1 Component segmentation and description of interfaces

2.4.1.2 Hardware/Software mapping

2.4.1.3 Management of persistent data

2.4.1.4 Access rights and access control

2.4.1.5 Global control flow

2.4.1.6 Tools

2.4.1.7 Reference

2.5 Modelling with UMLRT

2.5.1 Software architecture

2.5.1.1 Component segmentation and description of interfaces

2.5.1.2 Hardware/Software mapping

2.5.1.3 Management of persistent data

2.5.1.4 Access rights and access control

2.5.1.5 Global control flow

2.5.1.6 Tools

2.5.1.7 Reference

3. Glossary

CS - Chip Select

DIO - Digital In Output

FEE - Flash EEPROM Emulator

EA - EEPROM Abstraction

ECU - Electric Control Unit

EOL - End Of Line

HIS - Herstellerinitiative Software

ICU - Interrupt Capture Unit

MAL - Microconroller Abstraction Layer

MemIf - Memory Abstraction Interface

MCAL - Microconroller Abstraction Mayer

MCU - Microcontroller Unit

MMU - Memory Management Unit

Master - A device controlling other device.

NMI - Non Maskable Interrupt

OS - Operating System

Page - Smallest amount of memory that can be written in one pass.

PLL - Phase Locked Loop

PWM - Pulse Width Modulation

Sector - Smallest amount of memery that can be erased in one pass.

SFR - Special Function Register

Slave - A device beeing completely controllerd by a masted device.

RTE - Runtime Environment

WP - Work Package

STD Standard

REQ - Requirement

UNINIT Uninitialized

Bibliography

[1] **AUTOSAR GbR:** Requirements on memory services v. 2.2.2. Available from URL