AUTOSAR

Isabella Stilkerich

Friedrich-Alexander-Universität Erlangen-Nürnberg

Department of Computer Science 4
Distributed Systems and Operating Systems
Friedrich-Alexander University Erlangen-Nuremberg

http://www4.cs.fau.de/Research/KESO
www.autosar.org
http://www4.cs.fau.de/~isa
isabella.stilkerich@cs.fau.de
Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion
Motivation

- Climate Control Unit
- Infotainment
- Cruise control
- Navigation
- Fuel-injection
- Rain sensors
- Brake system
- etc.
AUTOSAR

- AUTOSAR development partnership founded in 2003
- Open standard for the E-/E-Architecture in the automotive field

Cooperate on standards – compete on implementation
Core partners and members

10 Core Partners

45 Associate Members

51 Premium Members

- BMW Group
- DaimlerChrysler
- PSA Peugeot Citroën
- Siemens VDO
- TOYOTA
- VOLKSWAGEN AG

- Fiat
- Honda
- Hyundai Kia Motors
- Infineon
- Mazda
- Porsche
- Philips
- Valeo
- TRW
- Magna

- Continental
- Bosch
- Delphi
- 3SOFT
- Telelogic
- DecoSys
- dSPACE
- ARM
- Infineon Technologies
- Mitsubishi Electric
- STI
- The MathWorks
- NEC Electronics Corporation
- Renesas
- Fujitsu
- Infosys
- Estrella
- Eta Automation Systems
- Dassault Systèmes
- Mentor Graphics
- Mentor Graphics
- Vector
- T-Systems

- Mentor Graphics
- Mentor Graphics
- Mentor Graphics
- Mentor Graphics
- Mentor Graphics
Benefits

- E/E-Architecture is divided into several layers
- Abstraction from underlying hardware
- Provision of Basic Software (BSW) Modules
- Integration of application modules from various suppliers
- Standardization of
  - Exchange formats
  - Interfaces
  - Methodology
Reusability of function in different vehicles

- Seat Control
- Light System
- Wiper System

Library

Configuration

Vehicle A

µ-Controller A
Distributed System A
Code

Vehicle B

µ-Controller B
Distributed System B
Code

Code
Virtual Function Bus (VFB)

- Strict separation of application and infrastructure logic
- The VFB concept is implemented by the Runtime Environment (RTE)
- The RTE provides an abstract “instruction set”
Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion
AUTOSAR Runtime Environment (RTE)

- System Services
  - ECU Tree Manager
  - Communication Manager
  - Event Handler Manager
  - Dispatcher Manager
  - Development Environment Manager
  - Synchronization Manager
  - Time Service

- Memory Services
  - NVRAM Manager

- Communication Services
  - COM
  - DCM
  - NM
  - IPDU
  - PDU Router
  - NM
  - TP

- I/O Hardware Abstraction
  - I/O Signal Interface
  - Driver for ext. ADC ASIC
  - Driver for ext. I/O ASIC

- Complex Drivers
  - I/O Drivers
    - CAN Driver
    - FlexRay Driver
    - USB Driver
    - UART Driver
    - PMM Driver
    - ADC Driver
    - DIO Driver
    - PORT Driver

- Onboard Device Abstraction
  - Wedflf

- Memory Hardware Abstraction
  - Mmflf
  - EA
  - Fee

- Communication Hardware Abstraction
  - xxx Interface
  - Trvc.
  - ext.Drv.

- Microcontroller Drivers
  - GPIO Driver
  - Watchdog Driver
  - MCU Driver
  - Core Test

- Memory Drivers
  - RAM Test
  - Internal Flash Driver
  - Internal EEPROM Driver

- Communication Drivers
  - SPI Handler Driver
  - LIN Driver
  - CAN Driver
  - FlexRay Driver

- Service Layer: OS (Operating System), I/O (Input/Output), etc.

- ECU (Electronic Control Unit) Abstraction Layer

- MCAL (Microncontroller Abstraction Layer)
Discipline “Software Architecture”

AUTOSAR SW-C 1
AUTOSAR SW-C 2
AUTOSAR SW-C 3
...
AUTOSAR SW-C n

AUTOSAR Runtime Environment (RTE)

System Services
- ECU/Node Manager
- Communication Manager
- Function Invocation Manager
- Diagnostic Event Manager
- Monitoring Manager
- Time Service
- Event Manager
- Centralized Resources

Memory Services
- NVRAM Manager

Communication Services
- COM
- DCM
- NM
- TP
- PDU
- IPDU

Onboard Device Abstraction
- Wdgt

Memory Hardware Abstraction
- Memif
- EA
- Fee

Communication Hardware Abstraction
- xxx Interface
- Trov
- ext.Drv

I/O Hardware Abstraction
- I/O Signal Interface

Complex Drivers
- Driver for ext. ADC ASIC
- Driver for I/O ASIC

Microcontroller Drivers
- GPT Driver
- Watchdog
- IMU Driver
- Core Test

Memory Drivers
- RAM Test
- Internal Pac Driver
- Internal EEPROM Driver

Communication Drivers
- SPI/Parallel Driver
- LIN Driver
- CAN Driver
- FlexRay Driver

I/O Drivers
- ICU Driver
- PWM Driver
- ADC Driver
- DIO Driver
- PORT Driver

Microcontroller
Overview

- Motivation
- AUTOSAR Layered Architecture
- **AUTOSAR Methodology**
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion
Methodology

- Methodologies are similar to cooking recipes, they describe
  - the "ingredients"
  - when and how these are used

- AUTOSAR defines requirements on
  - the software architecture
  - the software development methodology
SW-C Description

The description contains:

- Component interfaces
- Provided and required data and operations
- Hardware and infrastructure requirements
- Needs for services
- Information on specific implementation

Structure of information defined in the SW-C Template
ECU Description

- ECU = Electronic Control Unit (*German*: “Steuergerät”)
- Description of
  - Hardware elements such as processing units, memory
  - Hardware ports
  - Hardware connections
- Structure is defined by the ECU Resource Template
System Constraint Description

- Description contains:
  - Information on networking topologies (CAN, LIN, FlexRay)
  - Mapping and mapping constraints
  - Communication matrix
  - Protocols

- Structure defined by the System Template
System / ECU Configuration

- SW-Component Description
- ECU Resource Description (HW only)
- System-Constraint Description

AUTOSAR System Configuration Generator

Component API e.g. app.h

ECU Configuration Description

decisions (e.g. mapping)

ECU extract of System Configuration

decisions (e.g. scheduling, ...)

AUTOSAR ECU Configuration Generator

AUTOSAR RTE Generator

Generator for OS, COM, ...

Other Basic SW Generator

MCAL-Generator

Information / Database (no files)

Complex generation step: complex algorithm or engineering work

ECU Configuration Description

- RTE
  - Extract of ECU Config

- OS extract of ECU config e.g. OIL

- Basic SW Module
  - Basic SW Module A
  - Extract of ECU Config

- list of implementations of SW Components

per ECU
System Image

- ECU Configuration Description
  - RTE: Extract of ECU Config
  - OS extract: ECU config e.g. OIL
  - Basic SW: Module A extract of ECU config

- AUTOSAR RTE Generator
- Generator for OS, COM, ...
- Other Basic SW Generator
- MCAL Generator
- object files
- SW-C files
- source files
- Compiler
- Linker
- SW-C
- OS
- RTE
- MCAL
- other BSW
Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion
Software Components (SW-C)

- A SW-C needs to be described in an abstract way (SW-C description)
  - Interfaces
  - Internal behavior (e.g. runnable entities, events)
  - Implementation
- Source/object code
Interfaces (Classification)

- AUTOSAR Interface (relevant for SW-C)
- Standardized AUTOSAR interface
- Standardized interface

Interfaces: VFB & RTE relevant, RTE relevant, BSW relevant. Possible interfaces inside Basic Software (which are not specified within AUTOSAR).

AUTOSAR Interface

Standardized AUTOSAR Interface

Services

Communication

ECU Abstraction

Standardized Interface

Microcontroller Abstraction

Operating System

Standardized Interface

Applications

Standardized Interface

Application Software Component

Actuator Software Component

Sensor Software Component

AUTOSAR Runtime Environment (RTE)

Basic Software

ECU-Hardware

AUTOSAR Software

Application Software Component

AUTOSAR Interface

Interface

Standard Software
AUTOSAR Interface

- Interface=Typification of a connection between entities
- Definition of SW-C ports (interaction points=interface instance)
- Ports needed for SW-C Communication / Basic Software access
- Communication can be done locally or via a network
Interfaces and ports

- **Interface**
  - A type definition of an interaction point (port)

- **Port**
  - An instantiation of an interface.

- **Provided port**
  - A component offers an interface, that can be used by another component (created by providing component)

- **Required port**
  - A component needs access to an interface, that is provided by another component (created by the requiring component)

- **Required and provided ports have to be connected in a composition component**
**Sender-/Receiver-Interface**

- m:n communication, uni-directional
- Different semantics
  - Data distribution: Last-is-best (explicit/implicit)
  - Event distribution

![Diagram of Sender-SW-C, Receiver_1-SW-C, and Receiver_2-SW-C with Send/provide signal and receiver information paths.](attachment:image.png)
Sender-/Receiver-Interface (Example)

Sender-Receiver-Interface If_Switch {
  Boolean SwitchPosition
}

PositionSwitch
P_Switch

If_Switch
Client-/Server-Interface

- n:1 communication
- Synchronous and asynchronous calls
- Server runnable in
  - Task context
  - Client context

```
<interface>
Mem_Service
+ NvM_ReadBlock(void*, NvM_BlockIdType) : Std_ReturnType
+ NvM_WriteBlock(void*, NvM_BlockIdType) : Std_ReturnType
```

```
requests service
```

```
provides service
```

```
Client_1 SW-C
```

```
Client_2 SW-C
```
Interfaces and ports (revised)

- **Sender-/Receiver-Interface:**
  - Component sending the signal, defines the provided port – the receiver the required port

- **Client-/Server-Interface:** Different perspective
  - The direction of the control flow defines, which component has the provided or required port (function call direction)

- **Example:** SW-C_1 interested in sensor data by SW-C_2
  - SW-C_2 offers functionality, data flow from SW-C_2 to SW-C_1
  - Polling: SW-C_2 provides interface SensorValues

```
SensorValues
getExternalTemperature() : float
```
Interfaces and ports (revised)

- Direction of the control flow defines, which component has the provided or required port (function call direction)

- Example: SW-C_1 interested in sensor data by SW-C_2
  - SW-C_1 offers functionality, data flow from SW-C_2 to SW-C_1
  - Pushing: SW-C_1 provides notification interface SensorValuesUpdate

```
SensorValuesUpdate
```
Software Components (SW-C)

- A SW-C needs to be described in an abstract way (SW-C description)
  - Interfaces
  - Internal behavior (e.g. runnable entities, events)
  - Implementation
- Source/object code

![Diagram of SW-C components]

- C Type description
- C internal behavior description
- C implementation description
- Source / object code
Internal Behavior

Runnable Entities (runnables)
- Schedulable piece of code in SW-C
- Usually mapped to tasks, execution triggered by RTE

```c
void SwitchControl_CheckPosition() {
    Std_ReturnType success;
    boolean activated;
    ... /* Read HW state and write to activated */ ...
    success = Rte_Write_P_Switch_SwitchPosition(activated);
    ...
}
```

```c
TASK(Rte_Task) {
    ...
    /* trigger Runnable Entity Switch_Position */
    SwitchControl_Switch_Position();
    ...
}
```
Internal Behavior

- Events
  - Operation Invoked Event
  - Timing Event
  - Data Received Event
  - ...

- Points and accesses
  - Information about interaction between SW-C in Runnables

- Port API Options
  - Logical Representation of data entities on SW-C level
AUTOSAR SW-C vs Classical App

- No task bodies in SW-C
- SW-Cs communicate only via ports
  - No interrupt handlers in SW-C
  - No shared memory usage
- Abstract use of services via RTE
  - OS
  - Communication
  - Memory
  - I/O
Application Design
Application Design (Revised)
SW-C development (Summary)

- Model component (ports, interfaces)
- Connection between the SW-C
- Creation of a runnable, events
  - Entry in SW-C description
  - Needs on RTE (called cyclically, response to an event?)
  - or: mark as “can be invoked concurrently”
SW-C development (Summary)

- Model component (ports, interfaces)
- Connection between the SW-C
- Creation of a runnable, events
  - Entry in SW-C description
  - Needs on RTE (called cyclically, response to an event?)
- Writing source code for the SW-C
- Schedulable code resides in dedicated functions
SW-C development (Summary)

- Model component (ports, interfaces)
- Connection between the SW-C
- Creation of a runnable, events
  - Entry in SW-C description
  - Needs on RTE (called cyclically, response to an event?)
    - or: mark as "can be invoked concurrently"

- Writing source code for the SW-C
  - Schedulable code resides in dedicated functions

- Mapping of the function to an appropriate runnable
Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion
The Broker

Runtime Environment

OS

Services

I/O

Com

SW-C

SW-C

SW-C
Classification of Communication
- Intra-ECU com within a single SW-C (runnables)
- Intra-ECU com between SW-C
- Intra-ECU com between SW-C and BSW (system software)
- Inter-ECU communication

Implementation of communication patterns:
- Client-Server
- Sender-Receiver

Tasks of the RTE are also:
- Multiple instances of SW-Cs
- Synchronization (concurrency issues)

RTE code fully generated by the RTE generator
- Two-phase approach
**Example: Lost update issue**

- **Notice:** Can also occur, if REs have activated the `canBeInvokedConcurrently` feature, i.e. simple function call.
Synchronization

Developer sometimes has to ensure data consistency by
- Exclusive areas
- Inter-Runnable-Variables

Task_1

Rte_Enter()  Rte.Exit()
**Synchronization Techniques**

- Disable / Enable interrupts (via OS API)
- Sequential scheduling
- Task blocking
- Employment of OS resources
- Cooperative runnable placement (Fine-grained task blocking)
- Copy strategy
  - Read, write copy (Implicit S/R communication)
  - Single writer
- Per-Instance memory (SW-C developer has to insure consistency)

- No semaphores allowed in AUTOSAR!
Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion
AUTOSAR Basic Software (BSW)

- Communication Stack
- Memory Stack
- ECU State Management
- Complex Drivers
- System Stack
- I/O Stack
void Write_Value() {
    Std_ReturnType success;
    ...
    /* Write Value to Block */ ...
    success = Rte_Write_P_block_RAINDATA(55);
    ...
}
Memory Stack

Memory services:
- Abstract from location of memory devices
- Checksum protection and reliable storage

Memory Abstraction Interface: Broker

/* Memory Abstraction Interface */
Std_ReturnType MemIf_Write(
    uint8 DeviceIndex,
    uint16 BlockNumber,
    uint8* DataBufferPtr
)

/* Flash EEPROM Abstraction */
Std_ReturnType Fee_Write(
    uint16 BlockNumber,
    uint8* DataBufferPtr
)
Memory Interface

- Example 1: Single non-volatile device employed

```c
#define MemIf_Write (DeviceIndex, BlockNumber, DataBufferPtr) \
    Fee_Write (BlockNumber, DataBufferPtr)
```

- Example 2: Multiple device types employed

```c
/* MemIf.h */
extern const WriteFctPtrType WriteFctPtr[2];
#define MemIf_Write(DeviceIndex, BlockNumber, DataBufferPtr) \
    WriteFctPtr[DeviceIndex] (BlockNumber, DataBufferPtr)

/* MemIf.c */
#include "Ea.h" /* for getting the API function addresses */
#include "Fee.h" /* for getting the API function addresses */
#include "MemIf.h" /* for getting the WriteFctPtrType */
const WriteFctPtrType WriteFctPtr[2] = {Ea_Write, Fee_Write};
```
Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion
Conclusion

- Reduction of software complexity due to standardization
  - Costs
  - Time-to-market
- SW-C developers can concentrate on actual functionality
  - Model-driven approach
  - Faster development cycles
- Approved solutions for BSW and infrastructure
  - Code reusability
  - Reduce error-proneness
- Abstraction from Hardware facilitates portability
- Easier software module exchange
- Legacy software can be embedded in AUTOSAR architecture