Komponentenorientierte Automotive-Software-Entwicklung mit dem AUTOSAR-Standard

Dr. Heiko Dörr, Carmeq, Berlin
Kolloquium Automotive Software Engineering
TU Darmstadt, 12.05.2009
Content.

1. Model-based Development
2. Automotive System Engineering
3. AUTOSAR Standardization
4. Methodology of System Development
Model-based Development
Model-based Development.
General Task.
Model-based Development. Paradigm Shift.

Document based development process

Idea

Model based development process

Micro controller

A/D converter

Software component for computation of control functions

A/D converter

Software

D/A converter

Software component for computation of control functions

W_{INT}

R_{INT}

R_{DIS}

U_{DIS}

U_{INT}

- Idea
- Requirements / Behavioral model of system
- Physical model
- Implementation model
- Autocode

Micro controller

A/D converter

\( W_{\text{INT}} \)

\( R_{\text{INT}} \)

A/D converter

\( W_{\text{DIS}} \)

\( R_{\text{DIS}} \)

Software

D/A converter

\( U_{\text{INT}} \)

MbE-Slides by M. Conrad
Automotive Systems and SW Engineering
Automotive Open System Architecture
Cooperate on standards – compete on implementation
AUTOSAR Managing Complexity by Exchangeability and Reuse of Software Components

Exchangeability between supplier's solutions

Exchangeability between manufacturer's applications

Exchangeability between vehicle platforms
**Use case ‘Front-Light Management’**

**SwitchEvent**
- check_switch()
- switch_event(event)

**LightRequest**
- switch_event(event)
- request_light(type, mode)

**Front-Light Manager**
- request_light(type, mode)
- get_keyposition()
- set_light(type, mode)

**Headlight**
- set_light(type, mode)
- set_current(…)

---

**Operating System**
**Communication Control**
**Memory Management**
**Drivers**
**Hardware**

**ECU1**
Use case ‘Front-Light Management’
Exchange of type of front-light

Operating System
Communication Control
Memory Management
Drivers
Hardware

SwitchEvent
- switch_event (event)

LightRequest
- request_light (type, mode)
- get_keyposition()
- set_light(type, mode)

Front-Light Manager
- request_light(type, mode)
- get_keyposition()
- set_light(type, mode)

Xenonlight
- set_light(type, mode)
- set_current(…)

ECU2
Distribution on ECUs

SwitchEvent
- switch_event(event)
  AUTOSAR Int.

LightRequest
- request_light(type, mode)
  AUTOSAR Interface

Front-Light Manager
- request_light(type, mode)
  set_light(type, mode)
  AUTOSAR Interface

Xenonlight
- set_light(type, mode)
- set_current(…)
  AUTOSAR Interface
Use case ‘Front-Light Management’ – Multiple ECUs

SwitchEvent
  check_switch ()
  switch_event (event)

LightRequest
  switch_event(event)
  request_light (type, mode)

Front-Light Manager
  request_light(type, mode)
  get_keyposition()
  set_light(type, mode)

Xenonlight
  set_light(type, mode)
  set_current (…)

ECU3
  Operating System
  Communication Control
  Memory Management
  Drivers
  Hardware

ECU4
  Operating System
  Communication Cont.
  Memory Management
  Drivers
  Hardware

ECU5
  Operating System
  Communication Cont.
  Memory Management
  Drivers
  Hardware

CAN Bus
AUTOSAR Standardization
AUTOSAR – Core Partners and Members

Status: 13th March 2008

9 Core Partner

BMW Group
Continental
DAIMLER
PSA PEUGEOT CITROEN
TOYOTA
VOLKSWAGEN AG

52 Premium Member

Fiat Group Automobiles SpA
HONDA
KIA
HYUNID-KIA MOTORS
MAZDA
PORSCHE
MAN
VOLVO

General
OEM

Generic
Tier 1

Standard
Software

Tools and
Services

Semi-
conductors

Up-to-date status see: http://www.autosar.org
AUTOSAR
Project Objectives and Main Working Topics

- Implementation and standardization of basic system functions as an OEM wide “Standard Core“ solution
- Scalability to different vehicle and platform variants
- Transferability of functions throughout network
- Integration of functional modules from multiple suppliers

- Maintainability throughout the whole “Product Life Cycle“
- Increased use of “Commercial off the shelf hardware“
- Software updates and upgrades over vehicle lifetime
- Consideration of availability and safety requirements

Architecture
Application Interfaces
Methodology
Methodology:
Exchange formats or description templates to enable a seamless configuration process of the basic software stack and the integration of application software in ECUs and it includes even the methodology how to use this framework.

Application Interfaces:
Specification of interfaces of typical automotive applications from all domains in terms of syntax and semantics, which should serve as a standard for application software.

Architecture:
Software architecture including a complete basic or environmental software stack for ECUs – the so called AUTOSAR Basic Software – as an integration platform for hardware independent software applications.
Main Concepts: Architecture

- Basic Software modules
- Run time environment and communication
AUTOSAR ECU Software Architecture

Objectives:
- Basic SW: Decoupling of Hardware and Application Software
- Application SW: Relocation / Reuse of SW-Components between ECUs
AUTOSAR Basic Software Modules

**AUTOSAR Runtime Environment (RTE)**

- **System Services**
  - ECU State Manager
  - Diagnostic Event Manager
  - Watchdog Manager
  - Communication Manager
  - Development Error Manager

- **Memory Services**
  - NVRAM Manager
  - IPDU Router
  - DCM Diagnostic Com. Manager
  - CAN SM
  - CAN NM
  - FlexRay NM

- **Communication Services**
  - CAN Driver
  - LIN Driver
  - FlexRay Driver
  - CAN NM
  - FlexRay NM

- **Memory Hardware Abstraction**
  - EEPROM Abstraction
  - Flash EEPROM Emulation
  - Ext. EEPROM Driver

- **Microcontroller Drivers**
  - GPT Driver
  - MCU Driver
  - Watchdog Driver

- **Memory Drivers**
  - RAM Test
  - Internal EEPROM Driver
  - Internal Flash Driver
  - SPI Driver
  - SCI Driver
  - LIN Driver
  - CAN Driver
  - FlexRay Driver
  - MCU
  - μC
  - Flash
  - EEPROM
  - external EEPROM Driver

- **Communication Hardware Abstraction**
  - CAN Interface
  - LIN Interface
  - External CAN Interface
  - CAN NM
  - FlexRay NM

- **Onboard Device Abstraction**
  - Watchdog Interface
  - External Watchdog Driver

- **I/O Hardware Abstraction**
  - CAN NM
  - FlexRay NM

- **Device Drivers**
  - CAN Interface
  - LIN Interface
  - Ext. CAN Driver
  - Ext. LIN Driver
  - CAN NM
  - FlexRay NM

- **Communication Services**
  - CAN Driver
  - LIN Driver
  - FlexRay Driver

- **Memory Services**
  - NVRAM Manager
  - IPDU Router
  - DCM Diagnostic Com. Manager
  - CAN SM
  - CAN NM
  - FlexRay NM
Intra- and Inter-ECU Communication

- Ports implement the interface according to the communication paradigm (here client-server based).
- Ports are the interaction points of a component.
- The communication is channeled via the RTE.
- The communication layer in the basic software is encapsulated and not visible at the application layer.
1. AUTOSAR harmonizes already existing basic software solutions and closes gaps for a seamless basic software architecture.

2. AUTOSAR aims at finding the best solution for each requirement and not finding the highest common multiple.

3. The decomposition of the AUTOSAR layered architecture into some 40 modules has proven to be functional and complete.
Main Concepts: Application Interfaces

- Standardization approach
- Current stage of standardization
**AUTOSAR Application Interfaces**

**Syntax of Interfaces:**
- Meta-model, Software Component Template
- Supporting transferability within the network

**Semantics of Interfaces:**
- Physical properties, units, etc.
- Supporting re-use across product lines
- In scope of AUTOSAR work packages specifying application interfaces
To ease the re-use of software components across several OEMs, AUTOSAR proceeds on the standardization of the application interfaces agreed among the partners.

---

**Example**

<table>
<thead>
<tr>
<th>Data Type Name</th>
<th>LongAccBase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type Name</th>
<th>YawRateBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Yaw rate measured along vehicle z-axis (i.e. compensated for orientation). Coordinate system according to ISO 8855</td>
</tr>
<tr>
<td>Data Type</td>
<td>S16</td>
</tr>
<tr>
<td>Integer Range</td>
<td>-32768..+32767</td>
</tr>
<tr>
<td>Physical Range</td>
<td>-2.8595..+2.8594</td>
</tr>
<tr>
<td>Physical Offset</td>
<td>0</td>
</tr>
<tr>
<td>Unit</td>
<td>rad/sec</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

| Remarks               | This data element can also be used to instantiate a redundant sensor interface. Range might have to be extended for future applications (passive safety). |

<table>
<thead>
<tr>
<th>Data Type Name</th>
<th>RollRateBase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Major task: Conflict Resolution – Example Vehicle Speed

**Body Domain**
- **CentralLockingMaster**
- **InteriorLight**

**Powertrain Domain**
- **DriverReq**

**Chassis Domain**
- **CentralLockingMaster**
- **ESP**
- **ACC**
- **SSM**
- **EPB**
- **VLC**

Data Element: 
- **Name**: ActualVehicleSpeed
- **DataType**: VehicleSpeed
- **Min Bit size**: Uint12
- **Res**: 0.1
- **phys low**: 0.0
- **phys up**: 403.4 km/h

Data Element: 
- **Name**: ActualVehicleSpeed
- **DataType**: VehicleSpeed
- **Min Bit size**: Uint16
- **Res**: 0.00781
- **phys low**: 0
- **phys up**: 511.992 km/h

Data Element: 
- **Name**: VehicleLongSpeed
- **DataType**: VehicleLongitudinalSpeed
- **Min Bit size**: ??
- **Res**: ??
- **phys low**: ??
- **phys up**: ?? km/h

OK
For several domains a subset of application interfaces has been standardized to agreed levels.

It is a challenge to align standardization with the pace of application development.
Main Concepts: Methodology

- Overall methodology
- Structure of configuration information
- System Design – Implementation Process
Following the AUTOSAR Methodology, the E/E architecture is derived from the formal description of software and hardware components.

- Functional software is described formally in terms of “Software Components” (SW-C).
- Using “Software Component Descriptions” as input, the “Virtual Functional Bus“ validates the interaction of all components and interfaces before software implementation.
- Mapping of “Software Components” to ECUs and configuration of basic software.
- The AUTOSAR Methodology supports the generation of an E/E architecture.
**AUTOSAR Methodology**

**Derive E/E architecture from formal descriptions of soft- and hardware components**

VFB view

Standardized description templates for application software components (interfaces and BSW requirements)

Standardized exchange formats and methodology for component, ECU, and system level

Tools for
- support of component mapping
- generation of RTE, i.e. inter- and intra ECU communication

Standardized Basic Software (BSW) architecture, detailed specifications for implementation and configuration of BSW
To configure the system, input descriptions of all software components, ECU resources and system constraints are necessary.
The system configuration maps software components to ECUs and links interface connections to bus signals.
Input: Requirements & Vehicle Info

1a
SW Component Description

1c
System Description

2
Configure System & generate extracts of ECU descriptions

3
Configure each ECU

4
Generate SW executables for each ECU

1b
ECU Resource Description

Iterative corrections and/or optimizations (if required)
AUTOSAR – The Virtual Functional Bus
Input to the System Design on an abstract level

Example: speed warning device

- SW-Component-Description „get_v()“ describes a function to acquire the current vehicle speed and defines the necessary resources (such as memory, run-time and computing power requirements, etc.)
- Function „v_warn()“ makes use of „get_v()“
- „Virtual Integration“ by check of
  - completeness of SW-Component-Descriptions (entirety of interconnections)
  - integrity/correctness of interfaces
- The Virtual Functional Bus is implemented by the AUTOSAR-Runtime-Environment (RTE) and underlying Basic-SW

= tool based
AUTOSAR – Input Descriptions (1 of 3)
Step 1a): Description of SW-Components independently of hardware

Information for each SWC

- General characteristics (name, manufacturer, etc.)
- Communication properties:
  - p_ports
  - r_ports
  - interfaces
- inner structure (composition)
  - sub-components
  - connections
- required HW resources:
  - processing time
  - scheduling
  - memory (size, type, etc.)

SW Component Description

AUTOSAR-Description Editor

“get_v()” Software Component Description

= tool based
AUTOSAR – Input Descriptions (2 of 3)
Step 1b): Description of hardware independently of application software

Information for each ECU e.g. ECU1
- sensors and actuators
- hardware interfaces
- HW attributes (memory, processor, computing power, …)
- connections and bandwidths, etc.
- ...

ECU Resource Description

- General characteristics (name, manufacturer, etc.)
- Temperature (own, environment, cooling/heating)
- Available signal processing methods
- Available programming capabilities
- Available HW: - µC, architecture (e.g. multiprocessor)
  - memory
  - interfaces (CAN, LIN, MOST, FlexRay)
  - periphery (sensor / actuator)
  - connectors (i.e. number of pins)
- SW below RTE for micro controller
- Signal path from Pin to ECU-abstraction

= tool based
System Information
overall system
- bus systems, protocols,
  communication matrix and
  attributes (e.g. data rates, timing, …)
- function clustering
- function deployment
  (distribution to ECU)
- …

System Description

- Network topology
  - bus systems: CAN, LIN, FlexRay
  - connected ECUs, Gateways
  - power supply, system activation

- Communication (for each channel)
  - K-matrix
  - gateway table

- Mapping / Clustering of SW components

AUTOSAR – Input Descriptions (3 of 3)
Step 1c): Description of system

= tool based
AUTOSAR – System Configuration
Step 2: Distribution of SW-Component-Descriptions to ECU

- Configuration on the basis of descriptions (not on the basis of implementations!) of SW-Components, ECU-Resources and System-Description
- Consideration of ECU-Resources available and constraints given in the System-Description

AUTOSAR-System Definition (Distribution of SW-Component-Descriptions considering resources available)

SystemDescription
- e.g. mapping of signals to CAN matrices
- ...

ECU-Resource Description ECU1
- Description "get v()"
ECU-Resource Description ECU2
- Description "v_pwm()"
ECU-Resource Description ECU3
- SW Component 3
ECU-Resource Description ECU m
- SW Component n

Tool Based

an iterative process
**AUTOSAR – ECU-Configuration**

**Step 3: Generation of required configuration for AUTOSAR-Infrastructure per ECU**

- **Configuration-Descript. ECU1**
  - Description 1,
  - Description 2,
  - ...
  - Resources

- **System Description**
  - e.g. mapping of signals to CAN matrices
  - ...

**AUTOSAR - ECU Configuration Generator**

**AUTOSAR-RTE-Config-Info**
- communication mechanisms
- transport protocols
- ...

**AUTOSAR-Configuration ECU1**
- configuration of the AUTOSAR-RTE
- configuration of AUTOSAR OS
- configuration of MCAL
- Configuration of COM stack
- etc

= tool based
AUTOSAR – Generation of Software Executables

Step 4: Based on the configuration information for each ECU (example ECU1)
AUTOSAR Methodology – Conclusion

1. The E/E system architecture can be described by means of AUTOSAR.

2. The meta model approach and the tool support for specifying the AUTOSAR information model allow working at the right level of abstraction.

3. A methodology to integrate AUTOSAR software modules has been designed.

4. AUTOSAR pushes the paradigm shift from an ECU based approach to a function based approach in automotive software development.
Use case ‘Front-Light Management’ applying AUTOSAR

SwitchEvent
  check_switch()
  switch_event(event)
AUTOSAR Int.

LightRequest
  switch_event(event)
  request_light(type, mode)
AUTOSAR Interface

Front-Light Manager
  request_light(type, mode)
  get_keyposition()
  set_current(…)
AUTOSAR Interface

Xenonlight
  set_light(type, mode)
  set_current(…)
AUTOSAR Interface

Services

ECU Hardware

Can Driver

Microcontroller Abstraction

Std. Interface

AUTOSAR Interface

Standardized Interface

Communication

Std. Interface

Microcontroller Abstraction

ECU Hardware

Can Driver

DIO

AUTOSAR RTE

AUTOSAR RTE

AUTOSAR RTE
Further Information

http://www.carmeq.com

http://www.autosar.org

Published version of AUTOSAR Release 3.1

request@autosar.org