Entwicklung von eingebetteter Software in der Automobilindustrie mit AUTOSAR

Dr. Thomas Zurawka, SYSTECS, Februar 2015
Introduction

- The Vehicle – A driving low cost Computer network
  - ... more than 40 Mio. new vehicles with 1.000 Mio. ECU per year
  - ... ECU Software for 8...32-Bit Fixed-Point Processor
Introduction

- The **Vehicle** – A **driving low cost** Computer network contains
  - ... heterogeneous Components
    - Gasoline & Electrical Engines, Wireless Cell Phones, ...
  must be delivered
    - ... with 0 Defects
    - ... for 5,000 Euro – 150,000 Euro
    - ... within 3 years Development Time
  must be developed
    - ... together with many, highly integrated **Suppliers**
  must be maintained
    - ... for 30 years

Source for Throttle: Robert Bosch GmbH
Trends in the Automotive Industry

- The Vehicle - A driving **low cost** Computer network

  ![Controller](image)

  8 Bit … 32 Bit Controllers
  5kByte – 2MByte Bytes Flash

  *Why is there so much pressure on Controller size and Memory consumption?*

  ➔ *1MByte Flash costs 10 Euro!***
Trends in the Automotive Industry

- The Vehicle - A driving low cost **Computer network**
  - **Standardization** of ECU Software Architecture and Data due to problems with
    - ECU networks and
    - Exchange of Data **between Tools** and between OEM & suppliers
Trends in the Automotive Industry

- **The Vehicle** - A driving low cost **Computer** network
  - System & software engineers are using **Graphical Design**

... more than 60% of Application Software of an ECU graphically written

Tools must support co-existence of C and Graphic

Is C-Coding really necessary?
Trends in the Automotive Industry

- The Vehicle - A **driving** low cost **Computer** network
  - Road -> Test Cell -> Lab -> Math

Do we really need the Road for final Testing?
Trends in the Automotive Industry

- **Worldwide** distributed Development Teams
  - e.g. **1000 Employees & 100 Projects** per year for **one** ECU generation

_Do we already have the Tools which support this use case?_
Trends in the Automotive Industry

- **Increasing complexity**
  - An Example: Legal Requirements

![Graph showing emission regulations and zero emission concepts.](image)

- **Emission Regulations**
  - Euro 3
  - Euro 4
  - LEV
  - ULEV
  - SULEV

- **Real World**

- Zero Emission

- **CO [g/km]**
  - 0.0
  - 1.0
  - 2.0
  - 3.0
  - 4.0
  - 5.0
  - 6.0

- **HC [g/km]**
  - 0.0
  - 0.1
  - 0.2
  - 0.3
  - 0.4

- **Source:** Robert Bosch GmbH

Much more Software needed …

and …

other System Concepts, e.g. Hybrids, Fuel Cell, …
Trends in the Automotive Industry

- **Increasing Complexity** due to
  - Customer Requirements, Legal Requirements
  - more Variants and more complex Controllers
    - 50 Registers -> 1000 Registers

Which Tools can manage this huge amount of Data?
Trends in the Automotive Industry

- How does the Automotive Industry master these challenges?
  - CMMI or SPICE complaint Processes
  - Tricky development Methods and …
    - … Graphical Design & Test with Tools, Tools, Tools, …

Is CMMI 3, 4 or 5 the right final level?
Trends in the Automotive Industry

- **CAE-Tools** for the Development of **Software** for Vehicles
  - V-Cycle for ECU Software

Many Tools from different Vendors are in use!

Can **ONE Tool Family** or **ONE Vendor** dominate the V-Cycle?
Initiatives in the Automotive Industry

- The Virtual Development Initiative

Which Tool Vendor supports Co-Design of ECU Software, ECU Hardware and Vehicle Components first?

ECU Software  
ECU Hardware  
Throttle, …

Integrated Development Environment  
- ECU Simulation Backplane -
Initiatives in the Automotive Industry

- The Automation Initiative

Why is still a re-work of the Design Model necessary?

Why can we not generate the Design Model from the Requirements?

Automatic Code Generation

\[ a = b + c \]
Initiatives in the Automotive Industry

- The Automation Initiative

HIL Tests are very important for release of high quality Software!, but ...

With what language shall I code my Test Scripts?

Can I share them with others?

How can I manage them?

Is manual coding really necessary?
Initiatives in the Automotive Industry

- The Automation Initiative

- Automatic Test Script Generation

- Huge Efficiency increase possible, but...

- still no Technology available!
Initiatives in the Automotive Industry

The Standardization Initiative

- Standardization of Basic Software
- Transferability of Software Components throughout network
- Integration of Software Components from multiple suppliers

AUTOSAR Software Components

- AUTOSAR Runtime Environment
- Basic-Software (e.g. Networking, Flashing, Diagnostics, ...)
- Operating System
Initiatives in the Automotive Industry

- The Standardization Initiative

- Seat Heating - Supplier 1 -
- Window Lifter - Supplier 3 -
- Climate Control - Supplier 2 -

ECU 1 - Supplier 1 -
ECU 2 - Supplier 2 -
ECU 3 - Supplier 3 -
Part 1: AUTOSAR – in a Nutshell

Concepts – Software Component, Virtual Functional Bus, ECU Software Architecture
Overview

- This is the basic Idea
Overview

- ... now in AUTOSAR Terms
Basic Software

Application Software Layer

AUTOSAR Runtime Environment

Basic Software Layer

- System Services
- Memory Services
- Communication Services
- I/O Hardware Abstraction

- Onboard Device Abstraction
- Memory Hardware Abstraction
- Communication Hardware Abstraction

- Microcontroller Drivers
- Memory Drivers
- Communication Drivers
- I/O Drivers

Microcontroller

Operating System

Tools

standardized C-Code Module!

A/D, D/A, ...

EEPROM Driver

CAN, LIN, Flexray

Entwicklung von eingebetteter SW mit AUTOSAR © 2015 SYSTECS

© 2015 SYSTECS
Basic Software – The complete Picture (roughly 50 Modules)

- **System Services**
  - ECU State Manager
  - Communication Manager
  - Watchdog Manager
  - Diagnostic Event Manager
  - Function Inhibition Manager

- **Memory Services**
  - NVRAM Manager

- **Memory Hardware Abstraction**
  - Memory Abstraction Interface
  - EEPROM Abstraction
  - Flash / EEPROM Emulation

- **Communication Services**
  - AUTOSAR COM
  - Diagnostic Communication Manager
  - CAN Generic Network Mgmt
  - CAN Network Mgmt
  - FlexRay Network Mgmt
  - PDU Router
  - CAN Transport Protocol
  - FlexRay Transport Protocol

- **Communication Hardware Abstraction**
  - CAN Interface
  - FlexRay Interface
  - LIN Interface
  - CAN Transceiver Driver
  - FlexRay Transceiver Driver
  - CAN Network Mgmt

- **Microcontroller Drivers**
  - GPT Driver
  - Watchdog Driver
  - MCU Driver
  - RAM Test Driver
  - Flash Driver
  - EEPROM Driver
  - SPI Handler Driver
  - CAN Driver
  - FlexRay Driver
  - LIN Driver

- **I/O Drivers**
  - ICU Driver
  - PWM Driver
  - ADC Driver
  - DIO Driver
  - Port Driver

- **Development Error Tracer**
- **CRC Lib**

**Entwicklung von eingebetteter SW mit AUTOSAR** © 2015 SYSTEC
Part 2: AUTOSAR - Example

Introduction
An Example

- Measuring and Displaying the Tire Pressure
Part 2: AUTOSAR - Example

Step 1: Edit the Software Components
Step 1: The Software Components itself

- We define 3 Atomic Software Components
  
  - “Signal Adjustment”
  
  - “Tire Control”
  
  - “Display Info”
Step 1: The Ports of the Software Components

- The Software Components have to communicate, e.g. to exchange information
  - They need **Ports** to do this

- Ports will be assigned either to
  - **Sender-Receiver Interfaces** or
  - **Client-Server Interfaces**
    - Not used in this Example!

- We choose Sender-Receiver Interfaces to transport our Data
Step 1: The Ports of the Software Components

- We define the **Sender-Receiver Interfaces**
  - *PressureTemperature*
  - *Pressure*
  - *Warning*
  - *Calibration*
Step 1: The Ports of the Software Components

- We assign now the Sender-Receiver Interfaces to the Software Components
  - Ports become P-Ports or R-Ports
    - P-Ports provide information
    - R-Ports receive information
Part 2: AUTOSAR - Example

Step 2: Edit the Software Component Network
Step 2: The Software Component Network

- We define
  - A Composition “Vehicle Software”
Step 2: The Software Component Network

- We assign the Software Components to the Composition
Step 2: The Software Component Network

- We take the Composition "Vehicle Software" and connect the Software Components.
Step 2: The Software Component Network

![Diagram of software component network with labels for Signal Adjustment, Tire Control, and Display Info, connected by virtual functional bus.](image)
Part 2: AUTOSAR - Example

Step 3-1: Edit Runnables & Events for the Software Components
Step 3: Runnables & Events

- Where is the Code to fill the Software Components with Life?
- This is done with **Runnables**!
- For any Atomic Software Component you can define one or more Runnables
Step 3: Runnables & Events

- Runnables can be executed in the following ways
  - Periodically triggered with a defined Period
  - On Data Reception
    - with the Reception of a certain Data Element
    - The Data Element that should trigger the Runnable must be specified
What do we have so far?

Display Info
- Show Pressure
- Warning Request
- New Calibration
- Calibration
- Warning

Tire Control
- Pressure
- Temperature
- Calculate Warning
- New Calibration
- Calibrate

Signal Adjustment
- Send Values

Pressure
- Pressure
- Temperature

Send Values
- 100ms

Display Info:
- Pressure

Warning
- Calibration
- 500ms
- 100ms
Part 2: AUTOSAR - Example

Step 3-2: Edit the Code for the Software Components
Step 3: The Code for the Software Components

- We still miss the real Code!

- Based on the Information on the previous slide, **AUTOSAR Code-Generators** may generate Templates for each Software Component
  - Header Files (fully automatically generated)
  - C/C++ Files (only generation of Templates)
Step 3: The Code for the Software Components

- Here we are:

```c
/* Header File for Tire Control */

/* Output Interfaces */
RTE_WRITE_Pressure_Pressure (IN uint16 Pressure);
RTE_WRITE_PressureTemperature_Pressure (IN uint16 Pressure);
RTE_WRITE_PressureTemperature_Temperature (IN uint16 Temperature);
```
Step 3: The Code for the Software Components

- Here we are:

```c
/* Header File for Tire Control */
#include signal_adjustment.h

/* Runnable Entities */
Void Send_Values (void);
{
    uint16 sensor_temperature;
    uint16 sensor_pressure;

    Sensor_temperature = 20; /* That's trick, since we have no sensor software component */
    Sensor_pressure = 3 /* That's trick, since we have no sensor software component */

    RTE_WRITE_PressureTemperature_Temperature (sensor_temperature);
    RTE_WRITE_PressureTemperature_Pressure (sensor_pressure);
    RTE_WRITE_Pressure_Pressure (sensor_pressure);
}
```

Signal_Adjustment.c

Send Values

Pressure

Pressure

Temperature

100ms

Signal Adjustment
Step 3: The Code for the Software Components

- Here we are:

```c
/* Header File for Tire Control */

/* Input Interfaces */
RTE_READ_Calibration_Calibrate (OUT boolean* Calibrate);
RTE_READ_PressureTemperature_Pressure (OUT unit16* Pressure);
RTE_READ_PressureTemperature_Temperature (OUT unit16 Temperature);

/* Output Interfaces */
RTE_WRITE_Warning_Warning (IN boolean Warning);
```
Step 3: The Code for the Software Components

- Here we are:

```c
/* Header File for Tire Control */
#include tire_control.h

/* Runnable Entities */
Void Calculate_Warning (void);
{
    uint16 local_temperature;
    uint16 local_pressure;
    RTE_READ_PressureTemperature_Temperature (& local_temperature);
    RTE_READ_PressureTemperature_Pressure (& local_pressure);
    If (local_pressure < 3)
    {
        RTE_WRITE_Warning_Warning (1);
    }
}
```

Tire Control .c
What do we have so far?

Signal Adjustment

Pressure
Temperature
Send Values

100ms

Display Info

Pressure
Show Pressure
Warning Request
New Calibration

500ms

Warning
Calibration

Pressure
Temperature

Warning
Calibrate

Tire Control

Pressure
Temperature

Calculate Warning
New Calibration

100ms

Warning
Calibration
Step 3: Summary

Signal Adjustment
- Send Values
  - 100ms
  - Pressure
  - Temperature
  - Pressure

Tire Control
- Calculate Warning
- 100ms
- New Calibration
- Pressure
- Temperature
- Warning
- Calibration

Display Info
- Show Pressure
- Warning Request
- New Calibration
- Calibration
- Warning
- Pressure

Virtual Functional Bus

/* C File */
/* Header File */
/* C File */
/* Header File */
/* C File */
/* Header File */
Implementation of a Software Component

- Interaction Mechanisms
  - SWCs call RTE functions through a well-defined API
Implementation of a Software Component

- **Interaction Mechanisms**
  - The **RTE calls SWCs** by invoking user defined *Runnables*
  - This is simply a Call-back from the RTE to user code
Implementation of a Software Component

/* Runnable Entities */
Void <Runnable> (void);
{ RTE_WRITE_Port_Data (value); }

RTE Runnable callbacks:
MyRunnable();
...

RTE API calls:
RTE_SEND_Port_Event();
RTE_RECEIVE_Port_Event();
RTE_WRITE_Port_Data();
RTE_READ_Port_Data();
RTE_CALL_Port_Operation();
RTE_ENTER_Exclusive Area();
...

OS API calls:
ActivateTask();
SetEvent();
WaitEvent();
...

COM API calls:
COM_SendSignal();
COM_ReceiveSignal();
...

COM Callbacks:
COMCallback(...);
...

AUTOSAR SWC

<Runnable>

AUTOSAR Operating System

AUTOSAR COM

RTE

AUTOSAR Operating System

© 2015 SYSTECs
Implementation of a SW Component

- Example: Sensor Software Component

![Diagram showing the implementation of a sensor software component]

- Physical Interface
- Electrical Interface
- Electrical Interface

Temperature → Sensor → ECU Electronics → Microcontroller Peripherals → Hardware

Application SWC → Sensor SWC → Runtime Environment → Basic Software

- Algorithm
- Calculate T (I)
- Calculate I
- ADC_Read_Channel

I [mA] → U [V]

Sensor

Microcontroller

A/D converter

I/U converter

ADC Driver
Part 2: AUTOSAR - Example

Step 4-1: Edit the ECUs
Step 4: Definition of the ECUs

- For this Example we need **2 ECUs**:  
  - *Tire ECU*  
  - *Cockpit ECU*
Step 4: Definition of the ECUs

<<ECU Type>>
Tire ECU

<<Processing Unit>>
PU

<<Non-Volatile Memory>>
ROM1

<<Volatile Memory>>
RAM1

<<Communication Peripheral>>
CAN

<<ADC>>
I1

<<DIO>>
O1

<<Sensor Hardware>>
S1

<<Display Hardware>>
D1
Part 2: AUTOSAR - Example

Step 4-2: Edit the ECU Network
Step 4: Definition of a Network of ECUs

- Additionally we have to define a **Network** and we have to assign the 2 ECUs to it

- We use a CAN Network, called **High Speed CAN**

![Diagram of CAN Network with Tire ECU and Cockpit ECU connected to High Speed CAN.

<table>
<thead>
<tr>
<th>ECU</th>
<th>Bus Type</th>
<th>CAN Addressing Mode</th>
<th>Communication Speed</th>
<th>Protocol Name</th>
<th>Protocol Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tire ECU</td>
<td>CAN</td>
<td>Extended</td>
<td>500.000 Bits/s</td>
<td>-</td>
<td>2.0b</td>
</tr>
<tr>
<td>Cockpit ECU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part 2: AUTOSAR - Example

Step 5: Mapping Software Components to ECUs
Step 5: Mapping Software Components to ECUs

- Now we can map one or more Software Components to one ECU
What do we have so far?

Inter-ECU communication

Display Info
- Pressure
  - Show Pressure
  - Warning Request
  - New Calibration
- Warning
  - Calibration
- 500ms

Tire Control
- Pressure
- Temperature
- Warning
  - New Calibration
- 100ms

Cockpit ECU

Tire ECU

Send Values
- 100ms
- Signal Adjustment

What do we have so far?
Step 5: Summary (Just a redraw of the previous figure …)
Part 2: AUTOSAR - Example

Step 6: Edit Frames
Step 6: Frames for Inter-ECU communication

- Before any inter-ECU communication, e.g. external Connections can be configured we need to define the basic **Frames**
  - It contains a number of **Network Signals**
  - Define the structure of a Frame to be sent across the Network
  - But **not** the content of the Frame
Step 6: Frames for Inter-ECU communication

- Definition of a Frame
  - General Approach
Step 6: Frames for Inter-ECU communication

- We have 3 external connections
  - Pressure, Warning, Calibration
- So we define for each one a Frame (Communication Matrix)
Part 2: AUTOSAR - Example

Step 7: Map Data for Inter-ECU Communication
That was the Result of Step 5

- **Display Info**
  - Show Pressure
  - Warning Request
  - New Calibration

- **Tire Control**
  - Pressure
  - Temperature
  - Warning
  - Calibration
  - Calculate Warning
  - New Calibration

- **Signal Adjustment**
  - Pressure
  - Temperature
  - Send Values

- **Note**
  - uint8 pressure
  - Boolean warning
That was the Result of Step 6
That’s the Result of this Step 7
Part 2: AUTOSAR - Example

Step 8: Configuring the ECUs - Map Runnables to Tasks
Step 8: Task Mapping for Runnables

- The Runnables are triggered by Data Reception or Cyclic

- This is realized by an Operating System with
  - Tasks
  - Events
  - Alarms
  - ...

- We have to decide which Runnable is activated by which Task!
Signal Adjustment
Pressure
Temperature
Send Values
100ms
Pressure

Tire Control
Pressure
Temperature
Calculate Warning
100ms
Warning
Calibration
New Calibration

Display Info
Show Pressure
Warning Request
New Calibration
500ms
Calibration
Warning
Pressure

Operating System
Task A
Priority: 1
Task B
Priority: 2
Task C
Priority: 3

uint8 pressure
Boolean warning
uint8 pressure
Boolean Calibration
Part 2: AUTOSAR - Example

Step 9: Configuring the ECUs - Basic Software
Step 9: Configuring the Basic Software

- This step is necessary to setup a **real** ECU

- We have to configure the **Basic Software**
  - COM
  - PDU Router
  - CAN Interface
  - CAN Driver
  - …!
Entwicklung von eingebetteter SW mit AUTOSAR

© 2015 SYSTECS
Part 2: AUTOSAR - Example

Step 10: Code-Generation for ECUs
Step 10: Code-Generation for each ECU

- The AUTOSAR System for a Vehicle is stored as a **XML Description**

- The Configuration of the Basic Software for each ECU is stored as well as a **XML Description**

- **Code-Generators** generate now the complete Code for each ECU

- **Compiler & Linker** are generating the final Binaries!
Thank you for listening

SYSTECS Informationssysteme GmbH
Bahnhofstraße 11
D 70771 Leinfelden-Echterdingen

Phone +49- 711- 722 312 - 10
Fax +49- 711- 722 312 - 8

www.systecs.com

info@systecs.com