Semantic of execution in AUTOSAR

Pascal GULA,
C.E.O / pulse-AR
• Introduction to AUTOSAR

• AUTOSAR Software Architecture

• Semantic of execution of a SW-C

• Configuration of the OS and RTE

• Outlook on the future Timing Extension
Explosion of Electronic Content in Cars
Innovation through EE functions

1970

Electronic Injection
Check Control
Cruise Control
Central Locking
...
Needs that lead to AUTOSAR definition

**Increasing complexity of E/E systems**

- Increasing number and complexity of functions / ECU
- Complex interaction between functions (networking)

**The design of a vehicle becomes the design of a complete system** and does not consist solely with the assembly of functions

- Complex specification
- Low level of flexibility
- Increasing complexity of integration
Objectives (1/2)

- 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
- **Redundancy** activation
Hardware- and software will be **widely independent** of each other.

- Development processes will be simplified. This **reduces development time and costs**.
- Reuse of software increases at OEM as well as at suppliers. This **enhances** also quality and efficiency.

**Automotive Software will become a product.**
Technical Features

New concepts

Industry-wide consolidation of ‘existing’ basic software designs
Software Architecture Overview

AUTOSAR RTE:
- Standardized, openly disclosed interfaces
- HW independent SW layer
- Transferability of functions
- Redundancy activation

by specifying interfaces and their communication mechanisms, the applications are decoupled from the underlying HW and Basic SW, enabling the realization of Standard Library Functions.
Methodology Overview

Virtual Functional Bus

AUTOSAR SW-C

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

Mapping

Tool supporting deployment of SW components

Gateway

RTE
Basic Software

ECU I
AUTOSAR SW-C

ECU II
AUTOSAR SW-C

ECU m
AUTOSAR SW-C

...
AUTOSAR XML Description

Exchangeable and interoperable
Between AUTOSAR compliant tools and parties
Agenda

• Introduction to AUTOSAR

• AUTOSAR Software Architecture

• Semantic of execution of a SW-C

• Configuration of the OS and RTE

• Outlook on the future Timing Extension
Introductionary Use-Case – ECU Level (1/4)

- SwitchEvent
  - check_switch()
  - switch_event(event)
- LightRequest
  - request_light(type, mode)
- Front-Light Manager
  - request_light(type, mode)
  - get_keyposition()
  - set_light(type, mode)
- Headlight
  - set_light(type, mode)
  - set_current(…)

AUTOSAR Interface

- Standardized Interface
- Operating System
- Standardized Interface

- CAN Driver
- PWM
- DIO

ECU Hardware

- Microcontroller Abstraction
- ECU Abstraction
- Std. AUTOSAR Interface
- Std. AUTOSAR Interface
- Std. Interface

AUTOSAR RTE

- AUTOSAR Interface
- AUTOSAR Interface
- AUTOSAR Interface
- AUTOSAR Interface
- AUTOSAR Interface

Complex Device Drivers
Introductionary Use-Case – Changing Implementation (2/4)

- **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(…)`

- **AUTOSAR RTE**

- **AUTOSAR Interface**

- **Standardized Interface**

- **ECU Hardware**
  - **Microcontroller Abstraction**
  - **DIO**

- **Complex Device Drivers**

- **CAN Driver**

- **Services**

- **Communication**

- **Std. AUTOSAR Interface**

- **Std. Interface**

- **Operating System**

- **DIO**

- **AUTOSAR Int.**

- **Std. AUTOSAR Interface**

- **Std. Interface**

- **AUTOSAR Interface**

- **AUTOSAR Interface**
Introductionary Use-Case – Network Level (3/4)
Introductionary Use-Case – Network Level

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(…)

ECU
- Hardware

AUTOSAR RTE

AUTOSAR Interface

Std. AUTOSAR Interface

Services

Std. Interface

AUTOSAR Int.

Standardized Interface

ECU Abstraction

Std. Interface

Microcontroller Abstraction

ECU-Hardware

DIO

CAN Driver

Front-Light Manager

Std. Interface

Communication

Std. Interface

Standardized Interface

Microcontroller Abstraction

ECU-Hardware

CAN Driver

Xenonlight

Std. Interface

ECU Abstraction

AUTOSAR Interface

PWM

Std. Interface

Microcontroller Abstraction

ECU-Hardware

CAN Bus

Std. Interface

ECU-Hardware

CAN Driver
Components have two types of ports

- Provided ports
  - What the component give
  - This is done through *P-Ports*

- Required ports
  - What the component needs
  - This is done through *R-Ports*

Components can export as many P-Ports and R-Ports as needed
Software Component
- Interface Definition (2/2)

- Communication between SW-Cs is performed via SW-C ports
- Ports can be
  - Provided
  - Required
- Each port specify what it will communicate (using port interface)
  - Sender – Receiver Interface
    - Specify the **data** that will be sent / received
  - Client – Server Interface
    - Specify the **functions** that will be called / executed
  - Calibration Interfaces
    - Specify the values of **calibration** parameters
Software Component

- Composition Definition (1/2)
Compositions are built from:

- **SW-C Instances**
  - Called *Prototype*
- Assembly Connector instance
- Other Composition
  - A composition is hierarchical

All SW-C instantiated in a Composition must refer to a SW-C Type.
The purpose of mode is:
- To trigger runnables on the transition between modes
- To make runnables react differently

A particular SW-C:
- Manage modes and modes transitions
- Informs the others SW-C of modes transitions
Run-Time Environment Generation

RTE Contract Phase

- Generation of the RTE APIs
- Based on the SWC Component and Behavior
- Allows the SW-C development

RTE Generation

- Generation of the whole RTE
- Based on
  - All the previous SWCs present on a ECU
  - The configuration of the corresponding RTE
RTE Contract phase can occur as soon as SW-C are fully defined.

RTE Contract phase consists in:

- Generation of the specific APIs to access:
  - The Ports to send and receive data, explicitly or implicitly
  - The Ports to call and execute operations
  - The Callibration Port
  - The Inter Runnable Variables
  - The Exclusives Areas
  - The Per Instance Memories
  - ...

- These APIs are gathered in the ‘Component API’ (application header file).

RTE Contract phase allows:

- The SW-C development to be parallelised
- The SW-C developer to provide the SW-C’s source code without being concerned about the communication aspects.
RTE Generation - Contract Phase (2/2)

- RTE contract phase result is static
  - If the SW-C description change, the application header file must be regenerated

- RTE Contract phase result can be used as a “contract” between a OEM and a subcontractor

- SW-C description can be enhanced with the information from the specific implementation
  - This includes information about the memory needs for ROM and RAM
Software Component Mapping

- Two SW-Cs that exchange information on one ECU
  - The information can be handled ECU internally
    ⇒ We have created Intra-ECU communication

- Two SW-Cs that exchange information between different ECUs
  - The information must be handled ECU externally by the bus
    ⇒ We have created Inter-ECU communication
    ⇒ The AUTOSAR Code-Generator can generate this information automatically
ECU Basic Software
- Services view

AUTOSAR Software

SW Components

Basic Software

Application Software Component
  AUTOSAR Interface

Actuator Software Component
  AUTOSAR Interface

Sensor Software Component
  AUTOSAR Interface

AUTOSAR OS, RTE & Bsw Scheduler

AUTOSAR Memory Services

AUTOSAR Communication Services

AUTOSAR Network Management Services

AUTOSAR Complex Device Drivers Services
• Introduction to AUTOSAR
• AUTOSAR Software Architecture
• Semantic of execution of a SW-C
• Configuration of the OS and RTE
• Outlook on the future Timing Extension
Software Component - Behavior Definition

- Describe the real time execution elements and characteristics of an atomic SW-C
- Contains RunnableEntities to represent executable portion of code
- Contains RTE Event to trigger Runnables
- Provides communication scheme/concurrency scheme between runnables
- Provide Per Instance Memory dedicated to the instances of the associated SW-C
- Describes Service Needs with regards to underlying AUTOSAR Services
A Runnable belongs to a certain category

- Timing Event
- Data Send Completed Event
- Data Received Event / Data Receive Error Event
- Operation Invoked Event
- Asynchronous Server Call Returns Event
- Mode Switch Event / Mode Switched Ack Event
Runnable – Example Access

- DataWriteAccess
  - Specifies that a runnable implicitly sends a certain data element
    - Sending of data element values is only done once after runnable returns
    - Several usages of the API call inside the runnable cause only one data element transmission
  - Multiple DataWriteAccess can be declared for a runnable

- DataWriteAccess
  - A reference to the interface element that is sent
  - Used to build the API call
Implicit communication

Runnable A

Runnable B

This action is performed by the RTE at the start of runnable

Start runnable execution
Runnable job
Set data on port
Stop runnable execution

This action is performed by the RTE at the end of runnable

Start runnable execution
Get Data on port
Runnable job
Stop runnable execution
Explicit communication

**Runnable A**
- Start runnable execution
- Runnable job
  - Write data
  - ...
- Stop runnable execution

**Runnable B**
- Start runnable execution
- Runnable job
  - Read data
  - ...
- Stop runnable execution

**ASW-C 1**

**ASW-C 2**

Software Component
- Runnable Definition
Invoking an operation

- **ServerCallPoint**
  - Specifies that a runnable invoked an operation
  - Cannot be used concurrently
  - Can be a
    - AsynchronousServerCallPoint (Associated with AsynchronousServerCallReturnsEvent)
    - SynchronousServerCallPoint
  - Multiple ServerCallPoint can be declared for a runnable
Waiting

- Wait Point
  - Indicate that the Runnable is to wait for an \textit{RTEEEvent} to occurs
    - Contains a reference to all \textit{RTEEEvents} that can unlocked it
    - To stop infinite waiting, the call must specify a \textit{timeout}
  - A single Runnable can actually wait only at a single WaitingPoint
Client - Server synchronous communication

Software Component - Runnable Definition
Exclusive Area

- Allow to protect critical sections between runnables
  - If two or more Runnables refer to the same ExclusiveArea, only one is allowed to access it
- Two ways to use the ExclusiveArea
  - Entire Runnable runs in the Exclusive Area
  - Runnable dynamically enter and leave the Exclusive Area
    - Explicitly make API-calls to the RTE within the implementation of the RunnableEntity to enter and leave a specific ExclusiveArea
Inter Runnable Variable

- Support communication among runnables of the same component
  - Must have a data type
  - Can have a Initial Value

- Two different communication approaches
  - Explicit communication
    - Corresponds to DataReceivePoint/DataSendPoint
  - Implicit communication
    - Corresponds to DataReadAccess/DataWriteAccess
• Introduction to AUTOSAR

• AUTOSAR Software Architecture

• Semantic of execution of a SW-C

• Configuration of the OS and RTE

• Outlook on the future Timing Extension
AUTOSAR Runtime Environment (RTE)

Software Component A
Software Component B
Software Component Z

Operating System
BSW Scheduler

Access to OS services

The application part uses the OS services through the usage of RTE.

No direct access to OS services

The BSW modules use the OS services through the Bsw Scheduler.

Based on OSEK Time / OSEK OS
AUTOSAR OS, RTE & SchM: Configuration principles

- Two different domains: SWCs & BSWs
- Each domain will define its own tasks and execution contexts
Unique per ECU

- Is an interface layer between applicative parts (SW-C) & OS & BSW

- All configurations made in RTE are static configurations

- The XML models for SWC description allow an automatic generation of sources codes
  - Headers and source for RTE
  - Source template for application
AUTOSAR OS, RTE & Bsw Scheduler: RTE Communication Usage

- Implements communication between 2 SWC of a same ECU
- Implements communication between 2 runnables of a same SW-C
- Implements communication between 2 SWC between ECUs
OS Tasks Reminder

- Running
- Ready
- Waiting
- Suspended

Extended tasks only
Basic and extended tasks

1. Terminate
2. Activate
3. Release
4. Wait
5. Preempt
6. Start

Extended tasks only
Basic and extended tasks
Category 1A

- Runnable MUST terminate
- Contains at least an Implicit access and any Explicit Access
- Mapped to Basic/Extended Tasks
Category 1B

- Runnable MUST terminate
- Contains ONLY Explicit Access
- Mapped to Basic/Extended Tasks
– Category 2
  • May Contains Wait Point
  • Contains ONLY Explicit Access
  • Mapped mostly to Extended Tasks
  • Recommended only to map a Runnable to a Task (avoid some possible delays)
An ECU Parameter Definition contains a group of references to Module Definitions for a standard AUTOSAR ECU.

Each Module represents a BSW.

Each Module Definition defines the standard configuration parameters for a BSW.
ECU Parameter Definition
- OS Exemple

OS Module

OS Alarm
- AutoStart
  - AlarmTime
  - CycleTime
  - AppModeRef
- ? Action ?
  - ActivateTask
  - Callback
  - ...

OS Task
- Priority
- Activation
- Schedule
- ExecutionBudget
  - ...
- EventRef
- ResourceRef
- ...

OS Resource
- Property
- LinkedResource
ECU Parameter Configuration
- OS & RTE Example

**OS Module Configuration**

- **WakeUpTask**
  - Priority: 10
  - Activation: 1
  - Schedule: NON

- **CommandTask**
  - Priority: 2
  - Activation: 1
  - Schedule: NON

**RTE Module Configuration**

- **RunPowerManager**
  - PositionInTask: 1

- **RunStopLightsManager**
  - PositionInTask: 1

**StopLightsManager**

- **BehStopLightsManager**
- LightManagerWakeUp
- RunPowerManager
- RunStopLightsManager
- EventCommandStopLights
- BrakeStatus
• Introduction to AUTOSAR
• AUTOSAR Software Architecture
• Semantic of execution of a SW-C
• Configuration of the OS and RTE

• Outlook on the future Timing Extension
Introducing timing aspect on all steps of the AUTOSAR methodology:

- **VFB Timing**: this view deals with timing information related to the interaction of SwComponentTypes at VFB level.
- **SWC Timing**: this view deals with timing information related to the SwcInternalBehavior of AtomicSwComponentTypes.
- **System Timing**: this view deals with timing information related to a System, utilizing information about topology, software deployment, and signal mapping.
- **BSW Timing**: this view deals with timing information related to the BswInternalBehavior of a single BswModuleDescription.
- **ECU Timing**: this view deals with timing information related to the EcucValueCollection.
• Timing Notations are only considered at interface level (Atomic SW-C and Composition)
  – Time to execute some computation
  – Time to deliver a data
Timing Notations are now considered at behavior level

- Time to execute a runnable behavior (activation, start, termination)
Other Timing

• Timing Notations are then enriched for
  – System Timing: introduction of communication latencies (intra and inter-ECU)
  – BSW Timing: counterpart of SWC (runnables here are called entities)
• And aggregated for
  – ECU Timing: aggregate only ECU-relevant information from previous timing notation
• Basically, timing informations are:
  – Composed in a set of meta-model elements (TDEvents, TDEventChains, EventTriggering, LatencyTiming, Synchronization,...)
  – Referencing meta-model elements depending on the timing level previously described
• QUESTIONS ?